Become a Python Data Analyst

Python
数据分析师修炼之道

[美] 阿尔瓦罗 · 富恩特斯 著　　刘 璋 译



清华大学出版社

Python 数据分析师修炼之道

[美] 阿尔瓦罗·富恩特斯　著

刘　璋　译

清华大学出版社

北　京

内 容 简 介

本书详细阐述了与 Python 数据分析相关的基本解决方案，主要包括 Anaconda 和 Jupyter Notebook、NumPy 向量计算、数据分析库 pandas、可视化和数据分析、Python 统计计算、预测分析模型等内容。此外，本书还提供了相应的示例、代码，以帮助读者进一步理解相关方案的实现过程。

本书既可作为高等院校计算机及相关专业的教材和教学参考书，也可作为相关开发人员的自学教材和参考手册。

北京市版权局著作权合同登记号 图字：01-2019-1262

图书在版编目（CIP）数据

Python 数据分析师修炼之道/（美）阿尔瓦罗·富恩特斯（Alvaro Fuentes）著；刘璋译. —北京：清华大学出版社，2019

书名原文：Become a Python Data Analyst

ISBN 978-7-302-53016-9

Ⅰ. ①P…　Ⅱ. ①阿…　②刘…　Ⅲ. ①软件工具-程序设计　Ⅳ. ①TP311.561

中国版本图书馆 CIP 数据核字（2019）第 093947 号

责任编辑： 贾小红
封面设计： 刘　超
版式设计： 魏　远
责任校对： 马子杰
责任印制： 沈　露

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质 量 反 馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者： 北京富博印刷有限公司
装 订 者： 北京市密云县京文制本装订厂
经　　销： 全国新华书店
开　　本： 185mm×230mm　　**印　　张：** 8.5　　**字　　数：** 168 千字
版　　次： 2019 年 6 月第 1 版　　**印　　次：** 2019 年 6 月第 1 次印刷
定　　价： 69.00 元

产品编号：082450-01

译　者　序

当今，Python 已经成为一种主流的编程语言，它易于读写，非常实用，从而赢得了广泛的群众基础，被无数程序员热烈追捧。Python 几乎在每个领域都表现得非常优秀，这是一门真正意义上的全栈语言。

此外，Python 也是数据分析人员和统计人员在处理大量数据集和复杂数据可视化方面最常见和最流行的语言之一。具体来说，开发人员往往需要在工作中应用统计技术或数据分析，或者需要与 Web 应用程序进行交互。特别是，Python 在机器学习中的地位，它的机器学习库和灵活性的结合使得 Python 非常适合开发复杂的模型并可以直接在应用中加以使用。

本书介绍了 Python 语言中的核心工具和库，以帮助读者与数据分析处理过程协同工作、准备相关数据以执行简单的统计学分析，进而构建具有实际意义的数据可视化结果。本书将讨论 Python 语言中的各种库，例如 NumPy、pandas、matplotlib、seaborn、SciPy 和 scikit-learn，并将其应用于实际数据分析和统计示例中。

在本书的翻译过程中，除刘璋外，王辉、刘晓雪、张博、刘祎、张华臻等人也参与了部分翻译工作，在此一并表示感谢。

由于译者水平有限，难免有疏漏和不妥之处，恳请广大读者批评指正。

译　者

前　　言

Python 是高级数据分析师和统计人员所用的最常见和最流行的语言之一，可用于处理大型数据集和复杂的数据可视化任务。

本书介绍了 Python 语言中的核心工具和库，以帮助读者与数据分析处理过程协同工作、准备相关数据以执行简单的统计学分析，进而构建具有实际意义的数据可视化结果。本书将讨论 Python 语言中的各种库，如 NumPy、pandas、matplotlib、seaborn、SciPy 和 scikit-learn，并将其应用于实际数据分析和统计示例中。在阅读过程中，读者将会领略到如何高效地使用 Jupyter Noyebook，并借助于 NumPy 和 landas 库对数据进行操控。此外，还将利用 Python 库实现简单的预测模型、统计计算-分析和数据分析技术。

在阅读完本书后，读者在基于 Python 的数据分析方面将具备较为丰富的经验。

适用读者

本书面向初级数据分析师、数据工程师和 BI 专业人员，他们希望使用 Python 工具执行高效的数据分析。要理解本书所涉及的概念，读者应具备 Python 编程方面的一些背景知识。

本书内容

第 1 章：Anaconda 和 Jupyter Notebook。本章介绍了 Python 中一些较为重要的数据科学库，并对 Python 预测分析所用的主要对象、属性、方法和函数进行了整体描述。

第 2 章：NumPy 向量计算。本章讨论 NumPy 库，这也是 Python 项目中几乎全部科学计算所使用的库。学习如何使用 NumPy 数组，对于 Python 数据科学来说十分重要。

第 3 章：数据分析库 pandas。本章将整体介绍 pandas 库。对于 Python 编程语言来说，pandas 库提供了高性能、易于使用的数据结构和分析工具，因而受到了数据科学家以及 Python 社区开发者的喜爱。本章将通过相关示例展示如何利用 pandas 执行描述性分析。

第 4 章：可视化和数据分析。本章将考查数据科学的可视化效果。Python 针对不同的

功能提供了多种可视化选项。本章将学习两种最为流行的库，即 matplotlib 和 seaborn，并面向真实数据集执行探索性数据分析。

第 5 章：Python 统计计算。本章解释了如何利用 Python 执行统计计算，并据此考查包含青少年饮酒信息的数据集。

第 6 章：预测分析模型。本章简要介绍了预测分析，并通过构建一个模型对青少年的饮酒习惯进行预测。

资源下载

本书将引领读者整体了解 Python 中的数据分析过程、Python 数据科学栈中的主要库，并讨论如何使用各种 Python 工具有效地分析、可视化和处理数据。

读者可访问 http://www.packtpub.com 并通过个人账户下载示例代码文件。另外，在 http://www.packtpub.com/support 中注册成功后，我们将以电子邮件的方式将相关文件发与读者。

读者可根据下列步骤下载代码文件。

（1）访问 www.packtpub.com，利用电子邮件地址和密码登录，或注册。

（2）选择 SUPPORT 选项卡。

（3）单击 Code Downloads & Errata。

（4）在 Serach 文本框中输入书名。

当文件下载完毕后，确保使用下列最新版本软件解压文件夹。

- Windows 系统下的 WinRAR/7-Zip。
- Mac 系统下的 Zipeg/iZip/UnRarX。
- Linux 系统下的 7-Zip/PeaZip。

另外，读者还可访问 GitHub 获取本书的代码包，对应网址为 https://github.com/PacktPublishing/Become-a-Python-Data-Analyst。此外，读者还可访问 https://github.com/PacktPublishing/，以了解丰富的代码和视频资源。

下载彩色图像

另外，我们还进一步提供了本书所用截图/图表的彩色图像，读者可访问 http://www.

packtpub.com/sites/default/files/downloads/BecomeaPythonDataAnalyst_ColorImages.pdf 进行下载。

本书约定

本书通过不同的文本风格区分相应的信息类型。下面通过一些示例对此类风格以及具体含义的解释予以展示。

代码块如下所示。

```
# The largest heading
## The second largest heading
###### The smallest heading
```

当某个代码块希望引起读者的足够重视时，一般会采用黑体表示，如下所示。

```
[default]
exten => s,1,Dial(Zap/1|30)
exten => s,2,Voicemail

(u100)
exten => s,102,Voicemail(b100)
exten =>

i,1,Voicemail(s0)
```

图标则表示较为重要的说明事项。

图标则表示提示信息和操作技巧。

读者反馈和客户支持

欢迎读者对本书的建议或意见予以反馈。对此，读者可向 feedback@packtpub.com 发送邮件，并以书名作为邮件标题。若读者对本书有任何疑问，均可发送邮件至 questions@packtpub.com，我们将竭诚为您服务。若读者针对某项技术具有专家级的见解，抑或计划撰写书籍或完善某部著作的出版工作，则可访问 www.packtpub.com/authors。

勘误表

尽管我们在最大程度上做到尽善尽美，但错误依然在所难免。如果读者发现谬误之处，无论是文字错误抑或是代码错误，还望不吝赐教。对此，读者可访问 http://www.packtpub.com/submit-errata，选取对应书籍，然后单击 Errata Submission Form 超链接，并输入相关问题的详细内容。

版权须知

一直以来，互联网上的版权问题从未间断，Packt 出版社对此类问题异常重视。若读者在互联网上发现本书任意形式的副本，请告知网络地址或网站名称，我们将对此予以处理。关于盗版问题，读者可发送邮件至 copyright@packtpub.com。

目　　录

第 1 章　Anaconda 和 Jupyter Notebook

本书主要介绍基于 Python 的数据分析的基本概念。在第 1 章中，我们将学习如何安装 Anaconda，其中包含了本书所用的全部软件。此外，本章还将引入 Jupyter Notebook，这也是全部工作的计算环境。相应地，我们将通过具体解决方案帮助读者快速掌握相关工具。

本章主要涉及以下内容。

- Anaconda 及其所处理的问题。
- 如何在计算机设备上安装、启用 Anaconda。
- 通过 Jupyter Notebook 执行计算和分析任务。
- Jupyter Notebook 中一些有用的命令和快捷操作。

1.1　Anaconda

针对开发人员和数据科学家，Anaconda 是 Python 提供的一个免费、易于安装的包管理和环境管理工具，进而使得科学计算、数据科学、统计分析和机器学习中的包管理和部署更加简单。Anaconda 由 Continuum Analytics 推出，读者可访问 https://www.anaconda.com/download/免费下载。

Anaconda 是一个工具箱，即执行 Python 数据分析任务时的一个工具集。另外，读者也可免费下载独立的工具，但在后续操作中，获取整个工具箱则肯定更加方便。这也是 Anaconda 的用武之地——在查找各种工具并将其安装至系统的过程中，这将会节省大量的时间。除此之外，在单独安装 Python 包时，Anaconda 还负责处理所产生的包依赖关系，以及其他的潜在冲突和问题。

当访问 https://www.anaconda.com/download/时，可以看到针对不同操作系统的各种下载选项，读者需要根据自己的操作系统选取相应的安装程序。在下载页面中，读者将会看到两个安装程序，即 Python 3.7 和 Python 2.7，本书将采用 Python 3.7，如图 1.1 所示。

下载 Anaconda 软件的最新版本，并将其保存至 Downloads 文件夹中。

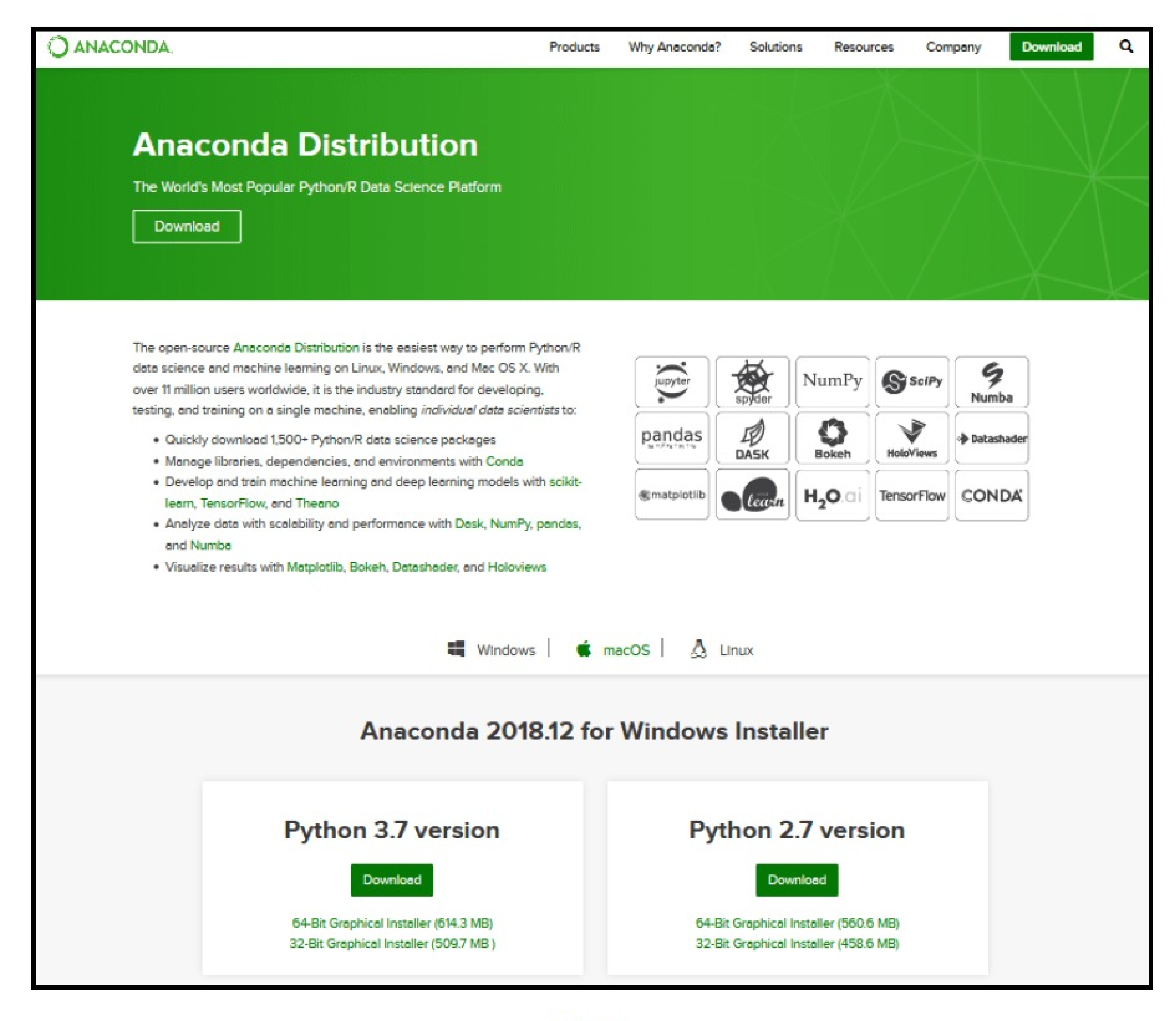


图 1.1

注意：

鉴于本书中的示例运行于 Windows 环境下，因而这里选择针对 Windows 的 64 位安装程序。macOS 和 Linux 的安装过程也基本类似。

Anaconda 的安装过程较为简单，且与其他软件的安装过程并无太多不同。双击.exe 文件，并在当前系统中安装 Anaconda 软件。相关步骤简单明了，另外，在软件的安装过程中，还会显示相应的提示信息。具体安装步骤如下。

（1）单击第一个安装程序对话框中的 Next 按钮。

（2）在浏览了软件的相关条款和条件后，单击许可协议中的 I Agree 按钮。

（3）在选项中选择 Just Me 并单击 Next 按钮。

（4）选取默认的安装目标文件夹并单击 Next 按钮。

（5）随后将询问环境变量，以及是否需要将 Anaconda 注册为默认的 Python。选中后单击 Install 按钮。

（6）安装结束后，单击安装程序对话框中的 Finish 按钮。

1.2　Jupyter Notebook

Jupyter Notebook 是一个 Web 应用程序，可创建、共享文档。该文档中包含了实时代码、等式、可视化结果以及解释性文本内容，其用途包括数据清理和转换、数值模拟、统计建模、机器学习等。Jupyter Notebook 类似于一个画布（Canvas）或环境，可使用编程语言（在当前示例中是 Python）执行计算并以非常方便的方式显示结果。

如果读者正在进行某种分析工作，那么 Jupyter Notebook 是非常方便的——其间通常会包含解释性文本、产生结果的代码和可视化结果，这些都显示在 Jupyter Notebook 中。据此，使用任何编程语言，特别是 Python，它都是一种非常方便的分析工作方法。Jupyter 项目诞生于 2014 年 IPython 项目。现在，它已经发展到支持多种编程语言的交互式数据科学和科学计算工具，因此可以将 Jupyter Notebook 与许多其他编程语言一起使用（多达 20 种语言）。Jupyter 这一名称来自 Julia、Python 和 R，这也是 Jupyter Notebook 最初支持的 3 种编程语言。

1.2.1　创建自己的 Jupyter Notebook

当启动 Anaconda 并开启 Jupyter Notebook 时，可从安装程序列表中单击 Anaconda Prompt。Anaconda Prompt 可视为一个终端（Terminal），用户可在其中输入相关命令。下面首先在桌面生成一个名为 PythonDataScience 的文件夹，该目录将存储在 Jupyter Notebook 中为本书编写和运行的所有 Python 代码。

一旦打开终端，可输入 cd Desktop/PythonDataScience 命令并按 Enter 键访问 PythonDataScience。为了启用该目录中的 Jupyter Notebook 应用程序，可输入 jupyter notebook 命令并按 Enter 键。这将启动当前应用程序，并可在浏览器的选项卡中看到该应用程序的主界面，如图 1.2 所示。

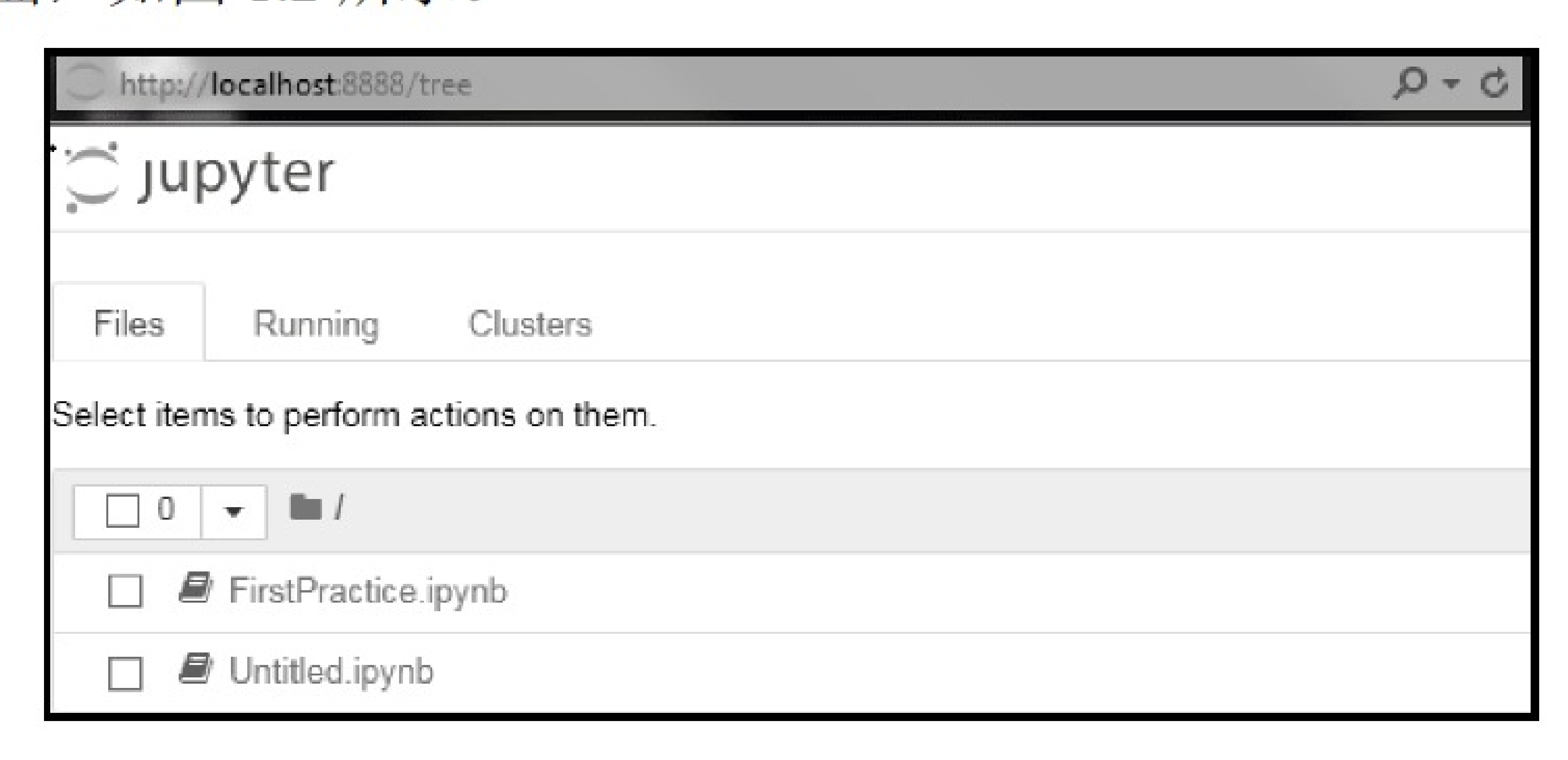


图 1.2

图 1.2 中包含了 3 个选项卡。在 Files 选项卡中，可以看到文件夹内包含的所有文件；在 Running 选项卡中，可以看到处于运行状态的程序，如 Terminal 或 Notebook；在 Clusters 选项卡中，则显示了与并行计算相关的细节内容，但本书将不会涉及这一特性。

Files 选项卡则是本书所用的主选项卡。当创建新的 Jupyter Notebook 时，可执行 New | Python 3 Notebook 命令，如图 1.3 所示。

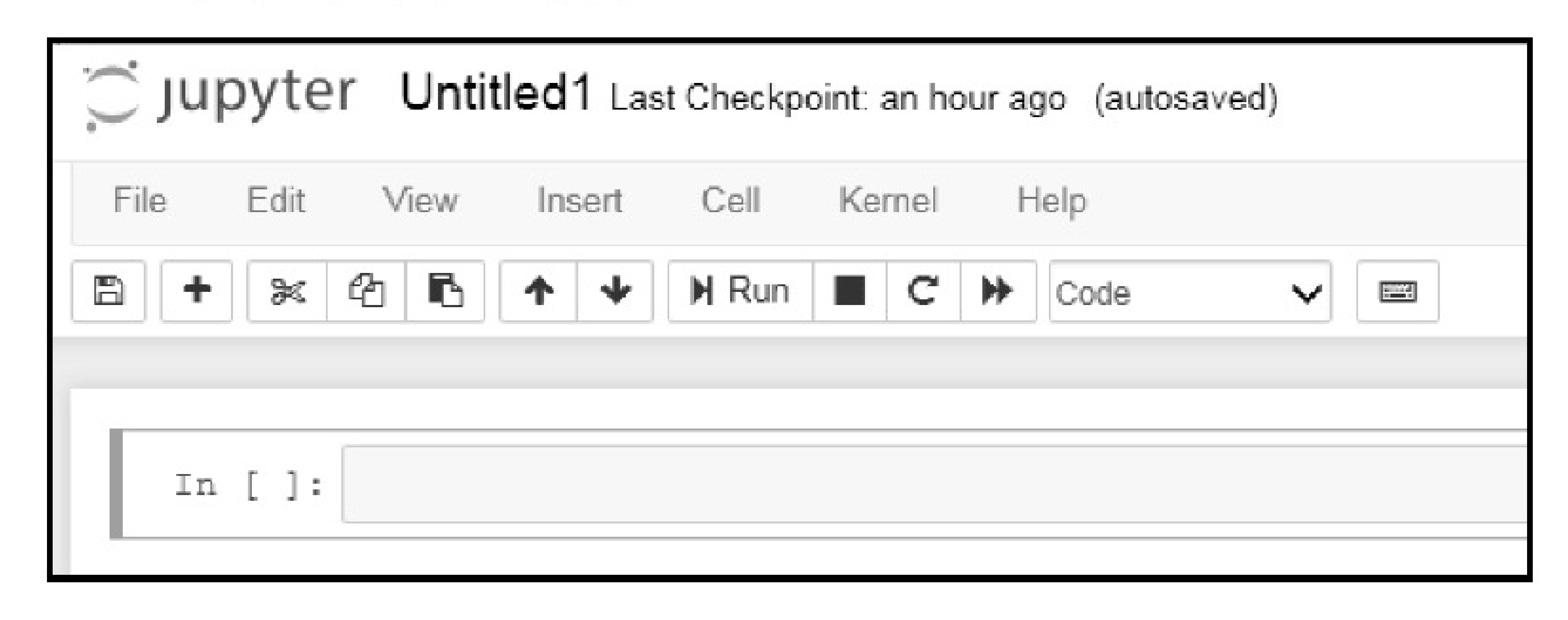


图 1.3

这将打开新的文件，即开始编码并运行 Python 代码的 Jupyter Notebook。

1.2.2　Jupyter Notebook 用户界面

Jupyter Notebook 包含了一些较为有用的界面，并可在操作过程中显示一些重要的信息和提示。此处访问 Help 命令，单击 User Interface Tour，并快速浏览一下 Jupyter Notebook 中的界面，如图 1.4 所示。

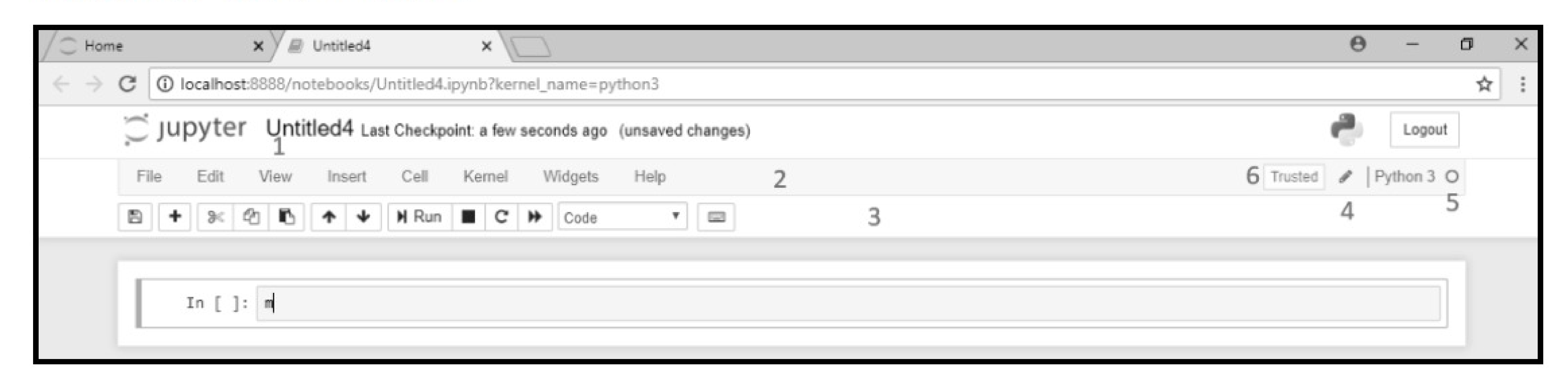


图 1.4

在 Jupyter Notebook 中的主页面中，包含了以下主界面。

- 标题（1）：表示为对应的文件名，用户还可对 Jupyter Notebook 的文件名进行修改。
- 菜单栏（2）：与其他桌面应用程序类似，菜单栏中包含了不同的操作。
- 工具栏（3）：位于菜单栏下方，其中包含了一些小图标，进而执行某些常见的

操作，如保存文件、分割单元、粘贴单元、移动单元等。

- 模式指示器（4）：位于菜单栏的右侧。Jupyter Notebook 包含了两种模式，即 Edit 模式和 Command 模式。其中，Command 模式中涵盖了许多可用的键盘快捷操作。在该模式中，指示器区域并不会显示任何图标，且需要对文件自身进行操作，如保存文件、复制和粘贴单元等。Edit 模式则允许用户在某个单元中编写代码或文本。当采用 Edit 模式时，将会在指示器区域看到一个铅笔图标。

注意：

Jupyter Notebook 由两种单元类型构成，即代码单元和文本单元。当在 Edit 模型下，所选单元的边框呈现为绿色。当从 Edit 模式返回 Command 模式时，可按 Esc 键或 Ctrl+M 快捷键。此外，还存在多种可用的键盘快捷方式，读者可访问 Help 命令查看快捷操作列表。

- 内核指示器（5）：显示系统的计算进程的状态。当中断进程中的计算时，可使用工具栏中的 stop 按钮。
- 消息区域（6）：该区域将显示相关消息，如 saving the file、interrupting the kernel 等。在消息区域中，用户可看到所执行的操作。

1.3　使用 Jupyter Notebook

下面打开新的 Jupyter Notebook，生成新的 Python 3 Jupyter Notebook，并将其命名为 FirstPractice。如前所述，Jupyter Notebook 由单元构成，其中包含了两种单元类型，即代码单元和文本单元。每次打开 Jupyter Notebook 时，将会显示代码单元，用户可于其中执行任何 Python 语句。

1.3.1　在代码单元格中运行代码

本节将尝试运行一些简单的代码语句，并学习如何在代码和文本间调整单元类型。对此，可在第一个代码单元格中输入 1+1 并执行该代码。当通过单击 run cell 按钮运行单元格中代码时，将会在代码单元格下方看到一行输出结果，如图 1.5 所示。

接下来生成一个变量 a，将其赋值为 10 并运行代码。虽然该变量已被创建，但考虑到尚未编写任何代码计算该变量，因而无法看到相应的输出结果。当执行这一条语句时，如果使用到该变量，例如，将该变量加 1，运行代码后将会看到如图 1.6 所示的结果。

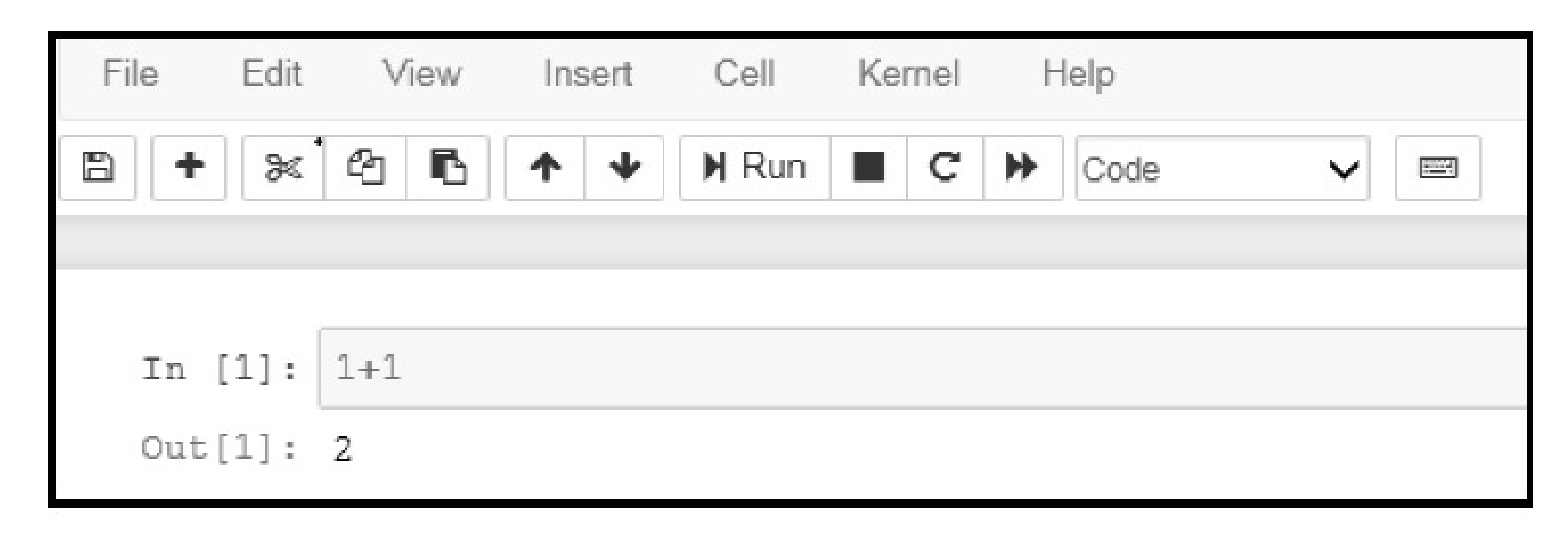


图 1.5

```
In [9]: a=10

In [10]: a+1

Out[10]: 11
```

图 1.6

下面考查基于变量 i 的 for 语法示例，如图 1.7 所示。

```
In [13]: for i in range(10):
             print(i)

         0
         1
         2
         3
         4
         5
         6
         7
         8
         9
```

图 1.7

当对应值位于 range(10)时，上述代码将通知 Jupyter Notebook 输出 i 值，对应结果如图 1.7 所示。

1.3.2　在文本单元格中运行 markdown 语法

之前曾谈到，单元格的默认类型是代码单元格，并可于其中编写 Python 表达式。除此之外，另一种类型则是文本单元格，并可用于编写文本内容，在当前输出结果下方的单元格中，可尝试输入 This is regular text。当执行 Cell | Cell Type | Markdown 命令时，即可通知 Jupyter Notebook，当前内容并非 Python 代码，而是文本内容。运行代码后，将会

发现输出结果表示为文本，且与我们输入的内容保持一致，如图 1.8 所示。



图 1.8

当采用 markdown 语法时，可通过多种方式格式化文本。如果读者对此感到陌生，可查看 Help | Markdown，即 GitHub 帮助页面。

注意：

Jupyter Notebook 采用的 markdown 与 GitHub 中的 markdown 保持一致。

文本的样式和格式十分丰富，读者可访问 https://help.github.com/articles/basic-writing-and-formatting-syntax/查看全部信息。本章仅讨论较为重要的标题内容。

当创建一个标题时，需要在标题文本前添加一个“#”符号（1～6 个“#”符号）。相应地，“#”符号的数量决定了标题的大小，其中，1～6 个“#”符号分别对应于最大和最小尺寸的标题，如下所示。

```
# The largest heading
## The second largest heading
###### The smallest heading
```

图 1.9 显示了上述语法的输出结果。



图 1.9

当在代码单元格中运行上述内容时，将会得到一系列的 Python 命令。如前所述，我们需要看到具有一定格式的文本内容，因而需要通知 Jupyter Notebook，通过将单元类型标记为 markdown，对应内容为文本。

1. 样式和格式

此外，我们还可尝试引入其他格式，如黑体、斜体、删除线以及黑体-斜体。图 1.10

显示了不同的样式及其对应的语法。

Style	Syntax	Keyboard shortcut	Example	Output
Bold	`** **` or `__ __`	command/control + b	`**This is bold text**`	**This is bold text**
Italic	`* *` or `_ _`	command/control + i	`*This text is italicized*`	*This text is italicized*
Strikethrough	`~~ ~~`		`~~This was mistaken text~~`	~~This was mistaken text~~
Bold and italic	`** **` and `_ _`		`**This text is _extremely_ important**`	**This text is *extremely* important**

图 1.10

另外，“>”符号指定了引用内容。尝试利用以下语法运行 markdown 单元。

```
In the words of Abraham Lincoln:
> Pardon my French
```

对应结果如图 1.11 所示。

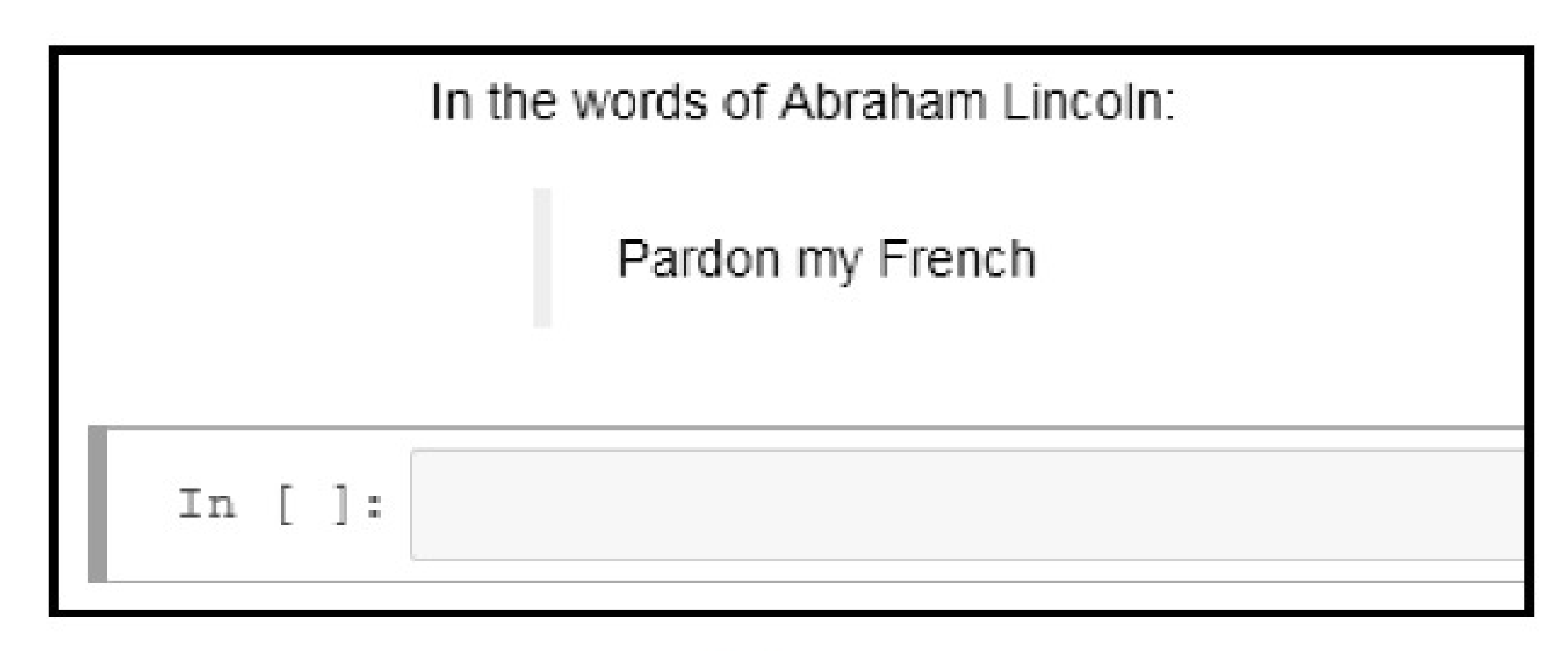


图 1.11

其中，文本的样式和格式保持一致，但引用的文本内容包含了一定的缩进。

2．列表

在一行或多行间使用“-”或“*”可创建项目列表形式。此外，还可使用 1、2、3 生成带有序号或有序的列表，如下所示。

```
- George Washington
- John Adams
- Thomas Jefferson
```

```
1. James Madison
2. James Monroe
3. John Quincy Adams
```

当运行上述 markdown 语法时，对应结果如图 1.12 所示。

- George Washington
- John Adams
- Thomas Jefferson

1. James Madison
2. James Monroe
3. John Quincy Adams

图 1.12

读者可访问 Jupyter Notebook 网站，其中包含了更为丰富的样式和格式语法。

1.3.3　键盘快捷操作

当每次需要运行单元格或者将代码单元格转换为 markdown 单元格时，频繁的鼠标操作使得整个过程异常烦琐。为了简化任务，Jupyter Notebook 中提供了大量可用的键盘快捷操作。下面考查较为重要的一些操作。

例如，当运行一个单元格时，如计算 1+2，则可使用 Alt+Enter 快捷键，该快捷方式将运行单元格中的代码，并在输出结果下方创建一个新的单元格。但是，如果仅希望运行计算过程，则可使用 Ctrl+Enter 快捷键，这使得 Jupyter Notebook 运行该单元格中的代码，并显示输出结果，但并不会生成新的单元格。如果打算在当前单元格下方插入新的单元格，则可按 B 键；按下 A 键则会在当前单元格上方创建新的单元格。当在 Jupyter Notebook 进行交互操作时，这将十分有用，其间需要生成多个单元格，进而包含更多的工作空间。

提示：

读者可通过 Help | Keyboard Shortcuts 命令来查看 Jupyter Notebook 中完整的快捷方式列表。

另一种较为有用的操作方式是在代码单元和 markdown 单元间进行转换。如果希望将代码单元转换为文本单元，可按 M 键，这将把 Edit 模式转换为 Command 模式。除此

之外，如果用户当前位于 Command 模式，且希望转换至代码单元或 Edit 模式，则可按 Esc 键。

1.4 本章小结

本章介绍了 Anaconda，同时还安装了本书所需的全部软件（包含于 Anaconda 中）。另外，我们还学习了 Jupyter Notebook 及其基本的操作方式。其中涉及代码单元、markdown 单元，以及与 Jupyter Notebook 协同工作时的一些快捷操作，以使工作流变得更加高效。

第 2 章将讨论 NumPy，它是执行数值计算时的核心库，同时也是 PythonX 系统中每个科学计算项目的核心内容。

第 2 章　NumPy 向量计算

本章将讨论 NumPy，这是一个 Python 编程语言库。此外，我们还将学习 NumPy 中较为重要的对象类型——数组，其间还会涉及数组的协同工作方式、重要的方法和属性。随后，本章还将应用所学的知识考查如何在实际操作过程中使用 NumPy。在本章的结尾，读者还将了解 Python 数据科学栈中的其他库，如 Matplotlib。相应地，本章将辅以相关示例进一步阐明 NumPy 的重要性及其所处理的问题。

本章主要涉及以下主题。

- ❑ NumPy 简介。
- ❑ NumPy 数组——构建、方法和属性。
- ❑ 基于数组的数学基础知识。
- ❑ 数组的操控方式。
- ❑ 通过 NumPy 执行模拟操作。

2.1　NumPy 简介

NumPy，也称作 Python 的向量化解决方案，是 Python 语言中执行科学计算的数据包。据此，我们可以创建多维数组对象；此外，相比于基本的 Python 功能，还可执行快速的数学计算。NumPy 是大多数 Python 数据科学生态圈中的基础内容。Python 中其他可用的数据分析库还包括 scikit-learn 和 pandas，且均依赖于 NumPy。以下内容展示了 NumPy 中的一些高级特征。

- ❑ NumPy 提供了一些较为高级的（广播）函数。
- ❑ 相关工具可与底层语言集成，如 C、C++和 Fortran 语言。
- ❑ 可执行线性代数以及复杂的数学计算，如傅里叶转换（FT）和随机数生成器（RNG）。

因此，如果希望执行某些大规模的高性能数据分析，且对运行速度有着较高的要求，则可尝试将 Python 代码与上述底层编程语言进行集成。

接下来通过一些示例讨论 NumPy 所处理的问题类型及其使用原因。假设对于一些与距离和时间相关的数据，需要对此进行计算以得到对应的速度。这里，一种方案是在 Python 中生成一个空表，编写 for 循环并写入结果，即距离除以时间，进而生成最终的速

度值，如图 2.1 所示。

```
In [1]: distances = [10, 15, 17, 26, 20]
        times = [0.3, 0.47, 0.55, 1.20, 1.0]

In [2]: # Calculate speeds with Python
        speeds = []
        for i in range(len(distances)):
            speeds.append(distances[i]/times[i])

        speeds

Out[2]: [33.333333333333336,
         31.914893617021278,
         30.909090909090907,
         21.666666666666668,
         20.0]
```

图 2.1

图 2.1 显示了经计算后的新的列表，即距离除以时间。当然，也可通过另一种更加 Python 化的方式执行列表推导。因此，如果使用[d/t for d,t in zip(distances, times)]，而非 for 循环，我们将会得到相同的结果，如图 2.2 所示。

```
In [3]: [d/t for d,t in zip(distances, times)]

Out[3]: [33.333333333333336,
         31.914893617021278,
         30.909090909090907,
         21.666666666666668,
         20.0]
```

图 2.2

对于给定的数据，这也是解决速度计算问题的传统方法，而之前的方法则是 Python 中所采用的基本方法。

下面根据购物分析查看另一个示例，也就是说，根据商品总量和每件商品的价格，计算购买总额。因此，为了得到全部购买总额，需要将每件商品数量乘以对应的价格；在乘法计算完毕后，需要再次执行加法运算。在 Python 中，一般通过以下代码予以表示。

```
sum([q*p for q,p in zip(product_quantities, prices)])
```

首先，代码中使用了一个 sum 函数，即数量乘以价格；随后，为了获得加法结果，可生成一个列表推导，这也是在 Python 中处理此类问题的常见做法。此时，如果运行该单元，对应结果为 157.1，即数量和价格集合中的全部总计结果，如图 2.3 所示。

```
In [4]: product_quantities = [13, 5, 6, 10, 11]
        prices = [1.2, 6.5, 1.0, 4.8, 5.0]
        total = sum([q*p for q,p in zip(product_quantities, prices)])
        total
Out[4]: 157.1
```

图 2.3

一种较好的方法是，通过 distances/times 获得速度值，并将第二个问题中的总量定义为 product_quantities 乘以 prices 列表，并得到乘法的求和结果。

然而，当运行单元代码时，将得到一条错误信息，其原因在于，Python 并不支持列表间的除法运算，以及列表间的乘法运算——但这正是 NumPy 的用武之地，即我们常说的向量化计算。此处，向量化是指对数组进行操作，如对象或列表，抑或逐元素执行操作。

2.2　NumPy 数组

NumPy 的主要对象是一个齐次多维数组。数组本质上是由相同类型的元素（通常是数字）组成的表，并通过正整数元组索引。在 NumPy 数组中，索引一般从 0 开始计数。因此，第 1 个元素为元素 0；第 2 个元素表示为元素 1，以此类推。在 NumPy 中，维度也称作轴（axe）；轴数或维度也称作数组的秩或维数。在将 NumPy 导入 Jupyter Notebook 中时，可使用 numpy as np 这一类导入语句。

2.2.1　在 NumPy 中创建数组

在 Python 中，以下两种方法可用于创建数组。

- 从列表中创建数组。
- 使用 NumPy 提供的内建函数。

1. 从列表中创建数组

当从列表中创建 NumPy 数组时，可使用 np.array 函数。前述示例中的列表，如 distances、times、product_quantities、prices，将通过 np.array 函数被转换为数组。对此，可针对每个列表运行以下代码行。

```
distances = np.array(distances)
times = np.array(times)
```

```
product_quantities = np.array(product_quantities)
prices = np.array(prices)
```

上述示例将把 Python 列表中的对象转换为 NumPy 数组。当考查这一新对象时，可在单元中调用 distances 对象名，进而运行该对象。随后将会看到如图 2.4 所示的数组。

```
In [9]:  distances
Out[9]:  array([10, 15, 17, 26, 20])
```

图 2.4

通过运行 type(distances)代码，还可查看该对象的类型。当在单元中运行上述代码时，将会看到如图 2.5 所示的输出结果，即对象类型。

```
In [10]:  type(distances)
Out[10]:  numpy.ndarray
```

图 2.5

其中，Python 显示了一个 numpy.ndarray 类型的对象。这里，n 表示为 n 维数组，也称作一维数组，有时可将其称为向量。

注意：

NumPy 向量和一维数组具有相同的含义。若传递 np.array()（即列表的列表），则会创建一个二维数组；如果传递一个列表的列表，将会生成一个三维数组；等等。

下面考查另一个示例，以便更好地理解如何通过一个列表的列表创建一个二维数组，如图 2.6 所示。

```
In [11]:  A = np.array([[1, 2], [3, 4]])
          A
Out[11]:  array([[1, 2],
                 [3, 4]])
```

图 2.6

这里，列表包含了两个元素，每一个元素都表示为外部列表中的列表本身，且每个列表均包含两个元素。因此，这可定义为一个包含两行、两列的二维数组。有时，也可将二维数组称作矩阵，即 2×2 矩阵。

2. 从内置的 NumPy 函数中创建数组

NumPy 自身也提供了一些函数可创建数组。某些时候，即使在未确定数组元素值的情况下，我们仍需要初始化数组。对此，有必要通过默认值生成数组。下面对这一类较为重要的函数进行逐一考查。

首先是 np.zeros 函数，该函数将生成包含 0 值的 NumPy 数组。作为可选项，还可指定数组的类型。在当前示例中，参数类型定义为 dtype=int，也就是说，数组的全部元素均为整型，如图 2.7 所示。

```
In [12]: # Create a length-10 integer array filled with zeros
         np.zeros(10, dtype=int)

Out[12]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

图 2.7

在上述示例中，np.zeroes(10, dtype=int)创建了一个包含 10 个整数的数组。最终，我们得到了包含 10 个 0 值的整型 NumPy 数组。

接下来是 np.ones 函数。作为可选项，可向该函数中传递 shape 函数，表示每个维度中元素的数量。这里，第一个维度中赋值为 3，第二个维度赋值为 5，参数类型定义为 float，如图 2.8 所示。

```
In [13]: # Create a 3x5 floating-point array filled with ones
         np.ones(shape=(3, 5), dtype=float)

Out[13]: array([[1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.],
                [1., 1., 1., 1., 1.]])
```

图 2.8

当运行 np.ones(shape=(3, 5), dtype=float)时，将得到一个 3×5 矩阵，且针对每个 float 类型对象填充 1。

NumPy 中还设置了一个非常有用的函数，可于其中创建数组并利用线性序列进行填充。相应地，存在以下两种方式可创建线性序列。

- 当采用 np.arange 函数时，需要提供一个起始点、结束点或数字，以及针对每次迭代的 step 值。如果未提供 step 值，则默认值为 1，如图 2.9 所示。

```
In [14]: # Create an array filled with a linear sequence
         # Starting at 0, ending at 20, stepping by 2
         np.arange(start=0, stop=20, step=2)

Out[14]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

图 2.9

当前示例调用了一个序列，其中，start 点为 0，stop 点为 20，step 值为 2。当运行单元时，将得到一个 NumPy 数组（起始于 0，step 值为 2）。该过程持续进行，直至到达结束点。需要注意的是，此处并未包含结束点或结束值。

- 另一种序列创建方法是使用 np.linspace 函数。对此，需要提供一个下限值、一个上限值，以及二者间均匀间隔的数值的数量。与图 2.9 中的 np.arange 函数不同，该函数中指定了相应的上限值，如图 2.10 所示。

```
In [15]: # Create an array of 20 values evenly spaced between 0 and 1
         np.linspace(0, 1, 20)

Out[15]: array([0.        , 0.05263158, 0.10526316, 0.15789474, 0.21052632,
                0.26315789, 0.31578947, 0.36842105, 0.42105263, 0.47368421,
                0.52631579, 0.57894737, 0.63157895, 0.68421053, 0.73684211,
                0.78947368, 0.84210526, 0.89473684, 0.94736842, 1.        ])
```

图 2.10

在当前示例中，我们得到一个 0～1 的序列（包括 1）；中间的所有值均是等距的，共计 20 个值。

2.2.2　数组的属性

NumPy 中的数组包含了多种属性，本节将考查 Python 中较为重要且经常使用的 3 个属性。对此，首先创建一个 float 类型的 3×4 数组，且全部值均为 1，如图 2.11 所示。

```
In [16]: A = np.ones(shape=(3, 4), dtype=float)
         A

Out[16]: array([[1., 1., 1., 1.],
                [1., 1., 1., 1.],
                [1., 1., 1., 1.]])
```

图 2.11

其中，第一个属性是维数，并可通过 A.ndim 属性查看数组的维度。在当前示例中，数组的维数为 2。对于 shape 来说，可采用 A.shape 属性获取每个维度中数值的数量。具体而言，第一个维度包含了 3 个数值，第二个维度中包含了 4 个值。最后，size 表示数组中所包含的全部元素数量。对此，A.size 属性表示数组中元素的总体数量，在当前示例中为 12。

2.2.3　数组中的基本数学运算

前面在介绍 NumPy 数组时曾讨论了列表中的 distances 和 times 值，下面通过数组方式执行一些较为基本的数学运算。如前所述，一种较好的方法是通过 distances 除以 times 这一方式计算 speeds。这可视为是一种向量化操作，并可通过 NumPy 数组予以实现。如果将 speeds 对象定义为 distances/times，NumPy 将逐个元素地执行该除法运算，即向量化操作，如图 2.12 所示。

```
In [17]: A.ndim
Out[17]: 2

In [18]: A.shape
Out[18]: (3, 4)

In [19]: A.size
Out[19]: 12
```

图 2.12

当运行单元格时，Numpy 将计算 10/0.3、15/0.47 等，进而获得 NumPy 数组中的速度向量。另一个操作示例则是计算 product_quantities 与 prices 之和。对此，可创建另一个名为 values 的数组，表示 product_quantities 乘以 prices 之和。随后，为了获取总额，可将 value 向量中的所有元素相加，如图 2.13 所示。

```
In [20]: speeds = distances/times
         speeds
Out[20]: array([33.33333333, 31.91489362, 30.90909091, 21.66666667, 20.        ])
```

图 2.13

当运行代码单元时，对应的结果值为 15.6（即 13*1.2）、32.5（即 5*6.5）等，这表示为向量 values，进而得到总计结果，也就是说，values 中所有元素之和。这一类操作 NumPy 予以实现，如图 2.14 所示。

下面考查更为复杂的例子，并创建另一个向量或 NumPy 数组。具体来说，x 中的 start 值为 0，stop 值为 20，step 值为 2，如图 2.15 所示。

```
In [21]: values = product_quantities*prices
         total = values.sum()
         print(values)
         total

         [15.6 32.5  6.   48.   55. ]

Out[21]: 157.1
```

图 2.14

```
In [22]: x = np.arange(start=0, stop=20, step=2)
         x

Out[22]: array([ 0,  2,  4,  6,  8, 10, 12, 14, 16, 18])
```

图 2.15

当运行代码单元时，将得到一组 0～18 的数字。当采用 NumPy 数组执行基本操作时，此类操作通常以逐个元素的方式进行。

例如，若向当前向量加 1，相应结果表明，原始数组中的每个元素均会加 1。类似地，当向该向量乘以 2 时，数组的每个元素均会乘以 2，如图 2.16 所示。据此，我们可执行诸如加、减、乘、除这一类基本的数学运算。

```
In [23]: x + 1

Out[23]: array([ 1,  3,  5,  7,  9, 11, 13, 15, 17, 19])

In [24]: x * 2

Out[24]: array([ 0,  4,  8, 12, 16, 20, 24, 28, 32, 36])

In [25]: x/2

Out[25]: array([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
```

图 2.16

除此之外，NumPy 还提供了一个通用函数，作为一类数学函数，该函数可在数组中加以使用，且以逐个元素方式执行。例如，如果需要知晓 x 中每个元素的正弦值，抑或每个元素的指数，则可执行 np.sin(x)和 np.exp(x)。当运行代码单元时，将得到如图 2.17 所示的结果。其中，sin 函数应用于每个元素上。类似地，指数函数也同样应用于每个元素上。

同样，我们也可以执行对数和合并计算。图 2.18 中显示了数组 x+1 的自然对数和平方根计算结果，且均以逐个像素的方式进行。

```
In [26]: # Universal functions
         np.sin(x)

Out[26]: array([ 0.        ,  0.90929743, -0.7568025 , -0.2794155 ,  0.98935825,
                -0.54402111, -0.53657292,  0.99060736, -0.28790332, -0.75098725])

In [27]: np.exp(x)

Out[27]: array([1.00000000e+00, 7.38905610e+00, 5.45981500e+01, 4.03428793e+02,
                2.98095799e+03, 2.20264658e+04, 1.62754791e+05, 1.20260428e+06,
                8.88611052e+06, 6.56599691e+07])
```

图 2.17

```
In [28]: np.log(x+1)

Out[28]: array([0.        , 1.09861229, 1.60943791, 1.94591015, 2.19722458,
                2.39789527, 2.56494936, 2.7080502 , 2.83321334, 2.94443898])

In [29]: np.sqrt(x)

Out[29]: array([0.        , 1.41421356, 2.        , 2.44948974, 2.82842712,
                3.16227766, 3.46410162, 3.74165739, 4.        , 4.24264069])
```

图 2.18

2.2.4　数组的常见操作

本节将考查 NumPy 数组的常见操作方式，其中涉及索引、切片（slice）和重构。

1．数组的索引

索引机制常用于获取、设置数组元素值。如果希望访问数组中的元素，则可像 Python 列表那样对其进行访问。为了进一步考查每个索引调用的执行方式，下面首先通过 np.linspace 生成一个一维数组，如图 2.19 所示。当访问数组的第一个元素时，对应的索引为 0。

```
In [59]: one_dim = np.linspace(-0.5, 0.6, 12)
         one_dim

Out[59]: array([-0.5, -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
                 0.6])

In [60]: one_dim[0]

Out[60]: -0.5
```

图 2.19

对于基于 0 值的索引，可通过 one_dim[0]运行代码单元，这将生成对应于索引 0 的元素。因此，在当前数组中，第一个元素为-0.5。当然，也可采用其他索引获取对应的元素值。另外，如果打算修改元素值，则可通过索引获取对应元素，并于随后执行赋值操作。例如，假设希望将索引为 0 的向量元素修改为 1，则可利用该索引获得元素并赋予新值。在图 2.20 中，第一个元素-0.5 被修改为 1。

```
In [32]: one_dim[0] = 1
         one_dim

Out[32]: array([ 1. , -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
                 0.6])
```

图 2.20

相应地，当与二维数组协同工作时，需要通过两个索引执行索引操作。下面首先创建一个二维数组。接下来，如果希望获取 0 行（索引为 0）的第 4 个元素值，即 3 列（索引为 3）中的第 1 个元素，则可使用 two_dim[0,3]，如图 2.21 所示。其中，第 1 个维度 0 表示为行索引，第二个维度 3 则表示为列索引。

```
In [33]: two_dim = np.array([[3, 5, 2, 4], [7, 6, 5, 5], [1, 6, -1, -1]])
         two_dim

Out[33]: array([[ 3,  5,  2,  4],
                [ 7,  6,  5,  5],
                [ 1,  6, -1, -1]])

In [34]: two_dim[0,3]

Out[34]: 4

In [35]: two_dim[0,0] = -1
         two_dim

Out[35]: array([[-1,  5,  2,  4],
                [ 7,  6,  5,  5],
                [ 1,  6, -1, -1]])
```

图 2.21

与一维数组操作类型，我们也可以修改二维数组中的任意元素值。例如，可将位置(0,0)中的元素值（即 0 行、0 列中的元素值）修改为-1，如下所示。

```
two_dim[0,0] = -1
two_dim
```

这将把对应的元素值从 3 修改为-1。

2. 数组切片

这里，切片（slice）是指在一个较大的数组中获取或设置一个较小的子数组，其操作方式与 Python 列表基本相同。为了更好地理解这一问题，此处再次考查运行 one_dim 代码时所使用的数组。据此，如果希望获取索引 0～5（不包括索引 5）的所有元素，则可运行 print(one_dim[2,5])代码单元。类似地，若打算获得前 5 个元素，则可采用 print(one_dim[:5])（不包括索引 5 对应的元素）。与 Python 列表类似，这里也可使用负索引。因此，如果希望获得最后 5 个元素，则可使用 print(one_dim[-5:])，如图 2.22 所示。

```
In [36]: one_dim

Out[36]: array([ 1. , -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
                 0.6])

In [37]: print(one_dim[2:5])
         print(one_dim[:5])
         print(one_dim[-5:])

         [-0.3 -0.2 -0.1]
         [ 1.  -0.4 -0.3 -0.2 -0.1]
         [0.2 0.3 0.4 0.5 0.6]
```

图 2.22

在二维数组示例中，实际规则并无变化，但需要指定每个维度上的切片。在当前示例中，如果希望得到数组中的前 4 个元素，则可使用 two_dim[:2,:2]。类似地，若打算获得全部行中起始的两个元素，可使用 two_dim[:,1:3]，如图 2.23 所示。

```
In [38]: two_dim

Out[38]: array([[-1,  5,  2,  4],
                [ 7,  6,  5,  5],
                [ 1,  6, -1, -1]])

In [39]: two_dim[:2,:2]

Out[39]: array([[-1,  5],
                [ 7,  6]])

In [40]: two_dim[:,1:3]

Out[40]: array([[ 5,  2],
                [ 6,  5],
                [ 6, -1]])
```

图 2.23

此处得到起始行和索引为 2 的行；对于列来说，情况也基本一致。类似地，图 2.23 中还显示了数组所有行中的前两个元素。

3. 重构数组

数组的重构是指将数组从一个维度调整至另一个维度，如一维数组至二维数组、一维数组至三维数组、三维数组至二维数组等。下面讨论如何对数组进行重构，此处将再次使用到前述一维数组。如果希望将数组重构或转换为一个 4×3 二维数组，则可使用 reshape()方法，如图 2.24 所示。

```
In [61]: one_dim

Out[61]: array([-0.5, -0.4, -0.3, -0.2, -0.1,  0. ,  0.1,  0.2,  0.3,  0.4,  0.5,
                0.6])

In [62]: one_dim.reshape(4,3)

Out[62]: array([[-0.5, -0.4, -0.3],
               [-0.2, -0.1,  0. ],
               [ 0.1,  0.2,  0.3],
               [ 0.4,  0.5,  0.6]])
```

图 2.24

当运行上述代码时，包含 12 个元素的一维数组被调整为 4 行 3 列的二维数组。除此之外，还可指定其他维度，如 2×6，甚至还可指定三维数组，如 2×2×3。

注意：

本书仅涉及一维和二维数组，因此读者不必担心更高的维度。

在二维数组示例中，若打算将其转换为一维数组，则可采用 flatten()方法，如图 2.25 所示。

```
In [43]: two_dim

Out[43]: array([[-1,  5,  2,  4],
               [ 7,  6,  5,  5],
               [ 1,  6, -1, -1]])

In [44]: two_dim.flatten()

Out[44]: array([-1,  5,  2,  4,  7,  6,  5,  5,  1,  6, -1, -1])
```

图 2.25

当运行 flatten()方法时，二维数组将被转换为一维数组。

2.3　使用 NumPy 进行模拟

本节讨论如何运用 NumPy 解决些实际问题，主要涉及两个模拟示例。其间我们还将学习与数组处理相关的其他操作。

2.3.1　投掷硬币

这里我们将使用 NumPy 模拟投掷硬币。为此，可使用 NumPy 随机子模块中的 randint 函数。该函数接收 low、high 和 size 参数，这也是希望输出的随机整数的范围。在当前示例中，输出结果为 0 或 1。相应地，low 值为 0，high 值为 2（但不包含 2）。另外，size 参数定义了希望输出的随机整数的数量，即投掷硬币的次数，对应代码如图 2.26 所示。

```
In [45]:  np.random.randint(low=0, high=2, size=1)
Out[45]:  array([0])
```

图 2.26

此处，0 表示硬币的正面，1 表示硬币的背面；另外，由于仅投掷一枚硬币，因而 size 值为 1。每次运行该模拟程序时，将会看到不同的输出结果。

下面考查另一种模拟情形，即一次投掷 10 枚硬币。相应地，需要将最后一个参数值调整为 size=10。为了得到硬币正面的总计结果，需要对 experiment 输出数组的所有元素求和，对应代码如图 2.27 所示。

```
In [45]:  np.random.randint(low=0, high=2, size=1)
Out[45]:  array([0])

In [46]:  experiment = np.random.randint(0,2, size=10)
          print(experiment)
          print(experiment.sum())

          [0 0 1 0 1 0 0 0 1 1]
          4
```

图 2.27

每次运行该模拟程序时，可以看到随机输出结果。假设希望执行该试验 10000 次，

这可通过 NumPy 方便地予以实现。下面创建一个 coin_matrix 模拟程序，并查看一次投掷 10 枚硬币时正面数量的分布状态。对此，我们将使用相同的函数 randint，以及相同的参数 0 和 2；但 size 将被赋予一个二维数组，因而有 size=(10000,10)。考虑到无法在屏幕上查看到 10000 行的矩阵，因而这里设置一个较小的矩阵显示 coin_matrix[:5,:]输出结果。

当运行该代码单元时，将会看到矩阵的前 5 行数据，且每次运行该模拟程序时，结果也将有所不同。

注意：

这里，前 5 行数据表示 10 枚硬币投掷 10000 次后（实际矩阵包含了 10000 行）的前 5 个结果。

当计算每次试验中硬币正面的次数时，可使用 sum 属性。但在当前示例中，我们希望对所有行求和。当在 NumPy 中对全部行执行求和计算时，可使用附加参数 axis，并设置 axis=1。这将生成一个数组，其中包含了每次试验中得到多少次正面的计数结果，如图 2.28 所示。

```
In [48]: counts = coin_matrix.sum(axis=1)
         print(counts[:25])
         print(counts.mean())
         print(np.median(counts))
         print(counts.min(), counts.max())
         print(counts.std())

         [5 4 7 8 6 2 7 5 6 8 6 4 6 6 2 5 6 5 6 3 3 4 6 7 6]
         4.9893
         5.0
         0 10
         1.5791724130062557
```

图 2.28

在图 2.28 中，我们获取了数组中的前 25 个元素，其中包含了每次试验中硬币正面的数量。此外，NumPy 还定义了一些数组，并涵盖了一些有用的方法执行统计计算，如均值、中值、最小值、最大值和标准偏差。当使用 mean()方法时，将得到全部试验结果的均值或硬币正面的平均值。median()方法将生成试验中硬币全部正面计数的中值。除此之外，还可使用 min()和 max()方法获得正面的最小和最大数量。std()方法将计算数组计数的标准偏差。

注意：

该试验每次的输出结果均有所不同，因此，读者不必为显示结果的差异而感到纠结。

如果希望了解硬币正面数量的分布状态，则可使用 bincount 函数。当运行代码单元时，将得到一个数值数组，表示试验中硬币的正面数量，且位于 0～10，如图 2.29 所示。

```
In [49]: np.bincount(counts)

Out[49]: array([  10,  109,  428, 1194, 2051, 2427, 2097, 1157,  431,   82,   14],
               dtype=int64)
```

图 2.29

此外，代码还包含了硬币正面数量分布的数值细节信息。可以看到，其中包含 0 次正面 10 次，1 次正面 109 次（以及对应的百分比），等等，如图 2.30 所示。

```
In [50]: unique_numbers = np.arange(0,11)
         observed_times = np.bincount(counts)
         print("===============\n")
         for n, count in zip(unique_numbers, observed_times):
             print("{} heads observed {} times ({:0.1f}%)".format(n, count, 100*count/2000

         ===============

         0 heads observed 10 times (0.5%)
         1 heads observed 109 times (5.5%)
         2 heads observed 428 times (21.4%)
         3 heads observed 1194 times (59.7%)
         4 heads observed 2051 times (102.5%)
         5 heads observed 2427 times (121.3%)
         6 heads observed 2097 times (104.8%)
         7 heads observed 1157 times (57.9%)
         8 heads observed 431 times (21.6%)
         9 heads observed 82 times (4.1%)
         10 heads observed 14 times (0.7%)
```

图 2.30

2.3.2　模拟股票收益

本节讨论来自金融领域的一个示例，并使用 matplotlib NumPy 库。假设我们想利用正态分布对股票的收益进行建模。这里，可使用 normal 函数生成具有正态分布的随机数。normal 函数中设置了 loc 参数、scale 参数（也称作标准偏差），以及加载随机值的参数。另外，random 参数表示营业年度的天数。

当运行代码单元时，将得到一个数值数组，表示前 20 天的收益。除此之外，也会得到一些负收益和一些正收益，就像在正常的股票市场中一样。假设 initial_price 为 100，当计算接下来几天中的全部价格时，可以使 initial_price 乘以收益累积和的指数函数。图 2.31 显示了基于 NumPy 的计算过程，以及相应的绘制结果。

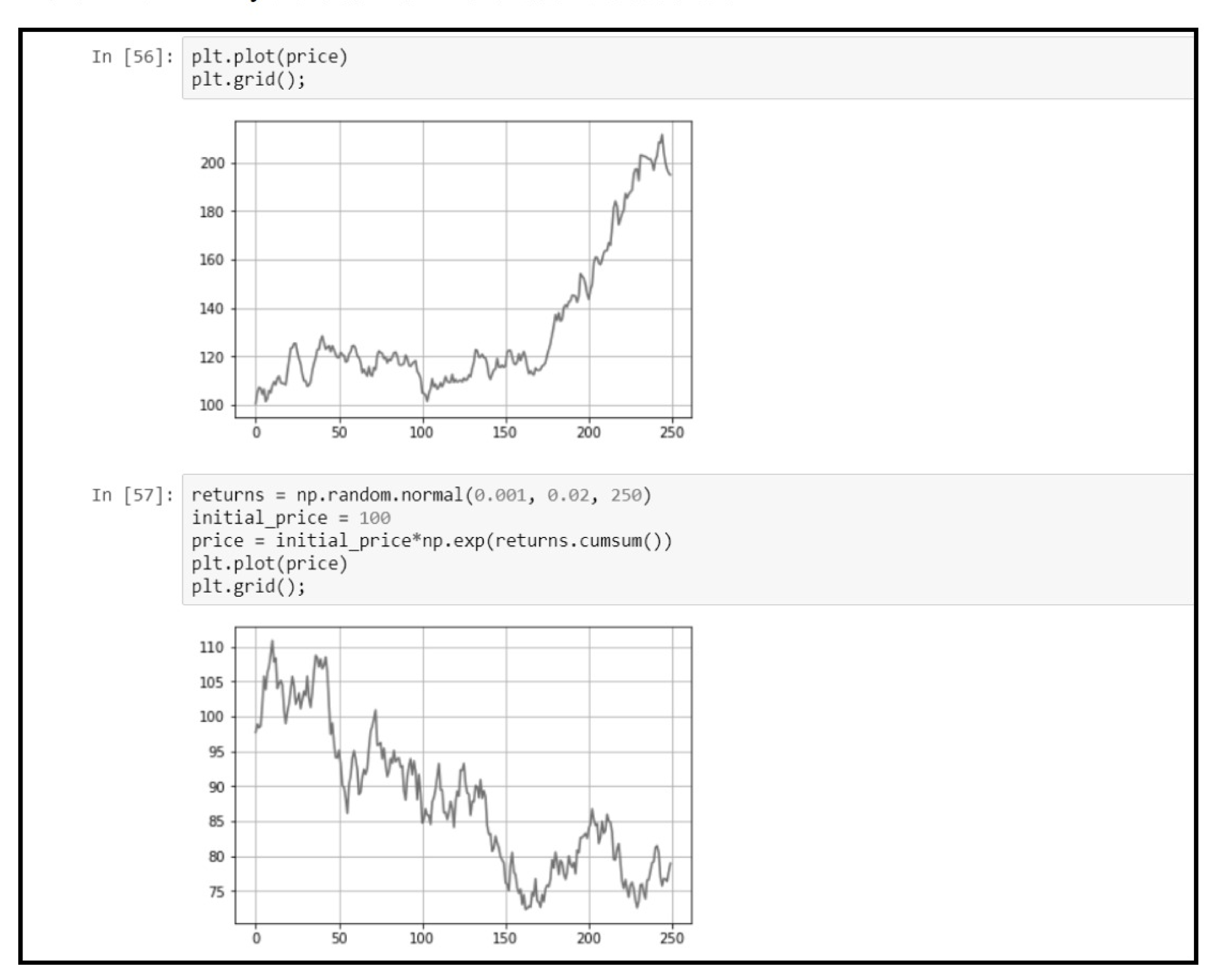


图 2.31

注意：

为了更好地理解上述程序，建议读者学习一点金融知识。当前目标并不是在金融领域中执行模拟过程，而是读者展示 NumPy 的简单性及其相关示例。

在当前示例中，股票价格始于 100，图 2.31 中模拟了其演变过程。对于相同的代码，每次运行代码单元时均会得到不同的模拟结果。

2.4　本章小结

本章讨论了 NumPy 库，旨在执行向量化操作。此外，我们还介绍了 NumPy 数组，这也是 NumPy 中的主要对象，其中涉及数组的构建、各种属性、基本的数学运算，以及对数组的操作方式。随后，本章还讲述了如何利用 NumPy 执行简单的模拟操作。

第 3 章将讨论 pandas，这也是 Python 中较为常用的数据分析库之一。

第 3 章　数据分析库 pandas

本章将讨论 pandas 库，涉及该库的一些功能，及其在 Python 数据科学中的重要性。此外，本章还将介绍 pandas 库中的主要对象，即 Series 和 DataFrame，包括其数据分析时的各项属性和操作。同时，我们还将考查相关示例，进而通过真实的数据集了解如何使用该库中的对象，并回答与数据集相关的一些简单问题。本章主要涉及以下主题。

- pandas 库。
- pandas 的操作。
- 通过示例回答一些与数据集相关的简单问题。

3.1　pandas 库

作为 Python 库，pandas 提供了快速、灵活的数据结构，并可与关系型或表格数据协同工作，如 SQL 或电子表格；对于真实的数据分析操作来说，pandas 也是一类较为基础的高层构建块。我们可通过下列语句导入 pandas 库。

```
#The importing convention
import pandas as pd
```

pandas 适用于以下场景。

- 表格数据中包含了不同类型的列，如 SQL 数据库或 Excel 电子表格中的数据。
- 有序或无序时序数据。
- 数据位于行和列中，其组织方式类似于矩阵。
- 使用观测数据或其他类型的统计数据。

pandas 中包含了以下两种主要的数据结构。

- Series：一维数据结构。
- DataFrame：二维数据结构。

它们可以处理不同领域中的大多数场景，如金融、统计学、社交科学，以及大多数工程和商业领域。pandas 构建于 NumPy 之上，并通过其他第三方库实现了与科学计算环境的良好集成。除此之外，我们还可将 pandas 库与其他库结合使用，如本章所讨论的可视化库。pandas 主要包含以下特征。

- 对于浮点和非浮点型数据，可以方便地处理遗失数据。
- 可方便地插入、删除 DataFrame 和更高维度对象中的数据。
- 数据自动对齐。
- 通过功能分组，实现功能强大、灵活的数据聚合和转换。
- 轻松地将不同索引的 Python 和 NumPy 数据结构转换为 DataFrame 对象。
- 基于标签的智能切片机制、良好的索引机制，以及大型数据集的子集划分。
- 较为直观的数据集的合并和连接机制。
- 轴的层次标记。
- 健壮的 IO 工具，用于从平面文件、Excel 文件、数据库和超高速 HDF5 格式保存/加载数据。
- 事件序列特定功能，包括日期范围生成和频率转换、移动窗口统计、移动窗口线性回归、日期变化、滞后等。

3.1.1　导入 pandas 中的对象

pandas 中包含了以下两种较为重要的对象。

- Series。
- DataFrame。

当利用 Python 并在数据科学计算中使用 pandas 时，首先需要导入 NumPy 和数学库，对应代码如下。

```
import numpy as np
import matplotlib.pyplot as plt
%maplotlib inline
```

接下来，当与 pandas 协同工作时，可利用标准的语句导入 pandas 库，如下所示。

```
import pandas as pd
```

3.1.2　Series

pandas 中的 Series 数据结构定义为一维标记数组，该数据结构主要包含以下特征。

- Series 中的数据可以是任意类型，如整型、字符串、浮点数、Python 对象等。
- 数据本质上是同质的，或者所有数据必须是同一类型的。
- 数据一般包含一个索引，进而生成如图 3.1 所示的数据结构字典和 Python 列表，

或者 NumPy 数组类型属性。

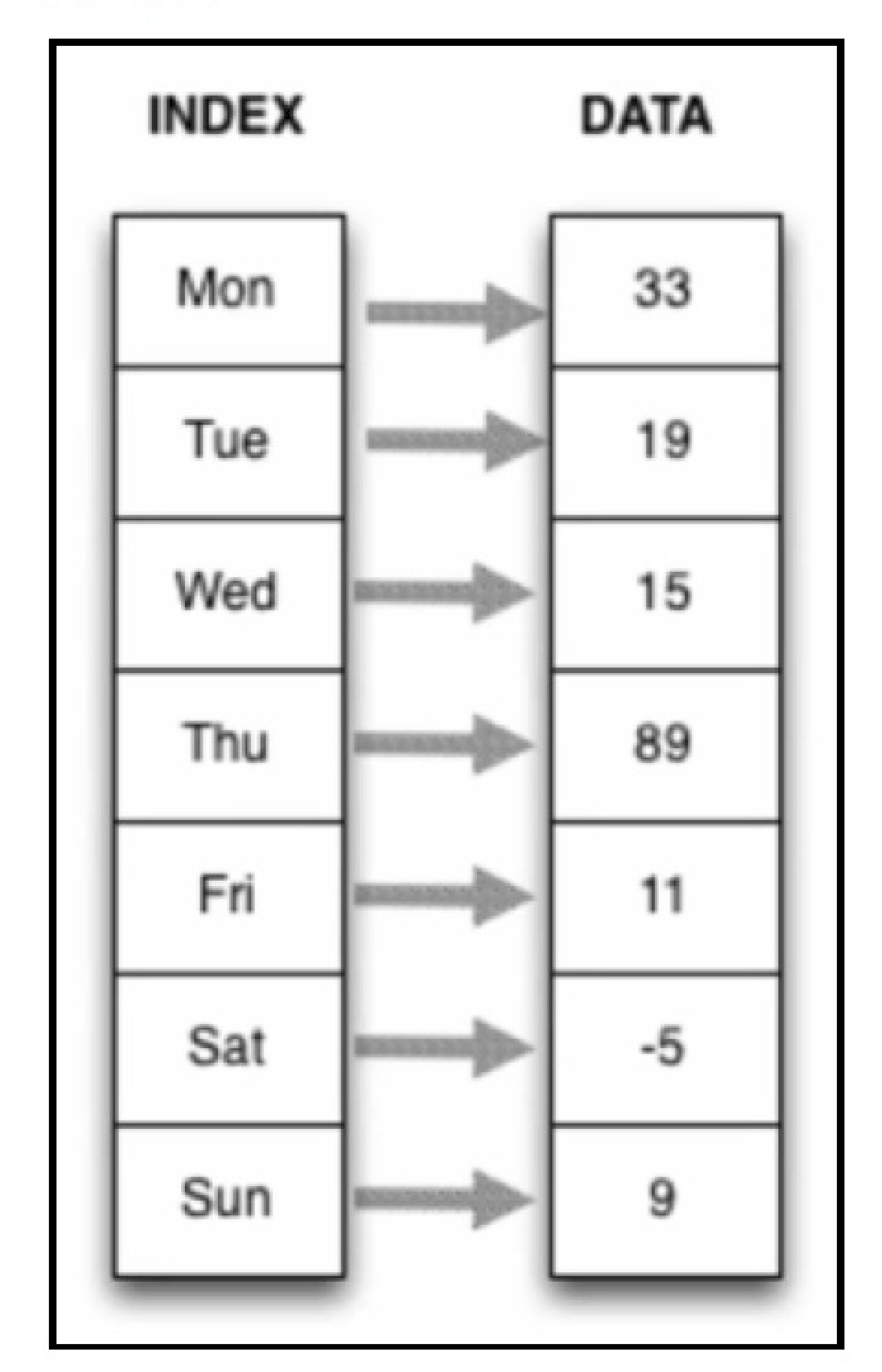


图 3.1

图 3.1 展示了 pandas Series 的可视化示例，可以看到，每个数据点均与一个索引关联。

3.1.3　创建 pandas 中的 Series

对此，存在多种方式可创建 pandas Series 对象。以下内容列举了较为常见的集中方法。

- ❑　从列表中进行创建。
- ❑　从字典中进行创建。
- ❑　从 NumPy 数组中进行创建。
- ❑　从外部数据源中进行创建，如一个文件。

下面考查如何从列表、字典和 NumPy 字典中创建 Series。对此，首先需要定义数据，并作为列表对其进行索引。此处生成了一个数值列表，并将其命名为 temperature；另一个数值列表则命名为 days。当从数据中创建一个 Series 时，可使用 pd.Series(temperature, index=days)构造方法，如图 3.2 所示。

```
In [3]: # define the data and index as lists
        temperature = [33, 19, 15, 89, 11, -5, 9]
        days = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']

        # create series
        series_from_list = pd.Series(temperature, index=days)
        series_from_list

Out[3]: Mon    33
        Tue    19
        Wed    15
        Thu    89
        Fri    11
        Sat    -5
        Sun     9
        dtype: int64
```

图 3.2

当运行上述代码时，可以看到，每个数值均与各自的索引进行关联。随后，将从 Python 字典中创建一个 Series。在 Python 字典中，一般包含了与对应数值相关联的键。因此，当从字典中生成一个 pandas Series 时，全部键将用作索引，而对应值则表示为与该索引所关联的 Series 中的数值。根据图 3.2 中所示数据，可定义一个 my_dict 字典，其中，日期（以星期几表示）将与温度关联。随后，将该字典传递至 pd.Series(my_dict)构造方法中，如图 3.3 所示。

```
In [4]: # from a dictionary
        my_dict = {'Mon': 33, 'Tue': 19, 'Wed': 15, 'Thu': 89, 'Fri': 11, 'Sat': -5, 'Sun': 9}
        series_from_dict = pd.Series(my_dict)
        series_from_dict

Out[4]: Mon    33
        Tue    19
        Wed    15
        Thu    89
        Fri    11
        Sat    -5
        Sun     9
        dtype: int64
```

图 3.3

虽然日期（以星期几表示）呈无序状态——Pyhton 字典中不包含隐含的顺序关系，但每一天均与自身的温度值相关联。例如，Fri 与 11 关联、Mon 与 33 关联等。

接下来讨论如何从 NumPy 数组中创建 pandas Series。对此，首先利用 np.linspace 函数定义 my_array 对象；随后，将该对象传递至 pd.Series 构造方法中，该方法根据所定义的 NumPy 数组创建一个 Series，如图 3.4 所示。

```
In [5]: # From a numpy array
        my_array = np.linspace(0,10,15)
        series_from_ndarray = pd.Series(my_array)
        series_from_ndarray

Out[5]: 0      0.000000
        1      0.714286
        2      1.428571
        3      2.142857
        4      2.857143
        5      3.571429
        6      4.285714
        7      5.000000
        8      5.714286
        9      6.428571
        10     7.142857
        11     7.857143
        12     8.571429
        13     9.285714
        14    10.000000
        dtype: float64
```

图 3.4

鉴于尚未指定任何索引，pandas 将自动生成整数索引，该索位于 0～元素数量减 1。当前示例中包含了 15 个元素，因而索引最大值为 14。

另外，还可通过 pandas Series 执行向量化操作，这与 NumPy 数组类似。具体来说，如果在 Series 上执行某项操作，那么，相同操作将应用于该 Series 中的每个元素，如图 3.5 所示。

在图 3.5 中，Series 乘以 2，也就是说，其中的每个元素均乘以 2。再次，代码将 Series 加 2，那么，其中的每个元素均会加 2。类似地，还可执行其他类型的数学运算，甚至可使用 NumPy 中的通用函数。对此，我们采用 NumPy 中的数学函数 np.exp 计算 Series 的指数。当运行代码时，将会看到包含相同索引以及最新指数值的 pandas Series。

```
In [59]: #Vectorized operations also work in pandas Series
         2*series_from_list

Out[59]: Mon     66
         Tue     38
         Wed     30
         Thu    178
         Fri     22
         Sat    -10
         Sun     18
         dtype: int64

In [60]: #Vectorized operations also work in pandas Series
         series_from_list + 2

Out[60]: Mon    35
         Tue    21
         Wed    17
         Thu    91
         Fri    13
         Sat    -3
         Sun    11
         dtype: int64

In [61]: #Vectorized operations also work in pandas Series
         np.exp(series_from_list)

Out[61]: Mon    2.146436e+14
         Tue    1.784823e+08
         Wed    3.269017e+06
         Thu    4.489613e+38
         Fri    5.987414e+04
         Sat    6.737947e-03
         Sun    8.103084e+03
         dtype: float64
```

图 3.5

3.1.4　DataFrame

DataFrame 定义为一种二维标记数据结构，且对应列可包含不同的数据类型。pandas DataFrame 类似于微软的 Excel 电子表格或 SQL 表。其中包含了两个索引，分别对应于行索引和列索引，如图 3.6 所示。

图 3.6 中包含了两个索引，这里，列与 Dates、Tokyo 等关联；而行则与 INDEX 进行关联。

INDEX	Dates	Tokyo	Paris	Mumbai
0	12-1	15	-2	20
1	12-2	19	0	18
2	12-3	15	2	23
3	12-4	11	5	19
4	12-5	9	7	25
5	12-6	8	-5	27
6	12-7	13	-3	23

图 3.6

3.1.5　创建 pandas DataFrame

DataFrame 包含多种构建方式，其中较为常见且重要的方法是从某个文件中创建 DataFrame。具体来说，可通过以下方式生成 DataFrame。

- ❑ 一维 ndarrays 字典、列表、字典或 Series。
- ❑ 二维 numpy.ndarray。
- ❑ TEXT、CSV、Excel 文件或数据库。

下面从真实的数据集中创建 DataFrame，对应的数据集涉及人力资源、人员流失、性能等方面的数据。读者可访问 https://www.ibm.com/communities/analytics/watsonanalytics-blog/hr-employee-attrition/以获取此类数据。

提示：

关于 pandas，读者可直接从互联网上下载相关数据。如果对应文件位于 URL 中，则可将该 URL 保存至 Python 字符串中，并利用 pandas 的 read_excel 函数直接创建 DataFrame。

考虑到将从 Excel 文件中生成 DataFrame，因而该过程会涉及一些具体的参数。其中，第一个参数是 io，并指定了文件的位置；第二个参数 sheetname 定义了要从 Excel 文件中读取的表；最后一个参数 index_col 则表示所使用的索引，如图 3.7 所示。

其中，定义了 file_url 变量表示文件的位置。随后，即可创建 DataFrame 并将其命名为 data。

```
In [7]: file_url = "https://community.watsonanalytics.com/wp-content/uploads/2015/03/WA_F

In [8]: data = pd.read_excel(io=file_url, sheetname=0, index_col='EmployeeNumber')
```

图 3.7

3.1.6 剖析 DataFrame

DataFrame 由以下 3 部分内容构成。

- 索引。
- 列名。
- 数据。

通过访问 index 和 columns 属性，可分别访问行和列标记，如图 3.8 所示。

```
In [9]: data.columns

Out[9]: Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
               'DistanceFromHome', 'Education', 'EducationField', 'EmployeeCount',
               'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
               'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
               'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'Over18',
               'OverTime', 'PercentSalaryHike', 'PerformanceRating',
               'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
               'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
               'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
               'YearsWithCurrManager'],
              dtype='object')

In [10]: data.index

Out[10]: Int64Index([   1,    2,    4,    5,    7,    8,   10,   11,   12,   13,
                     ...
                     2054, 2055, 2056, 2057, 2060, 2061, 2062, 2064, 2065, 2068],
                    dtype='int64', name='EmployeeNumber', length=1470)

In [11]: data.values

Out[11]: array([[41, 'Yes', 'Travel_Rarely', ..., 4, 0, 5],
                [49, 'No', 'Travel_Frequently', ..., 7, 1, 7],
                [37, 'Yes', 'Travel_Rarely', ..., 0, 0, 0],
                ...,
                [27, 'No', 'Travel_Rarely', ..., 2, 0, 3],
                [49, 'No', 'Travel_Frequently', ..., 6, 0, 8],
                [34, 'No', 'Travel_Rarely', ..., 3, 1, 2]], dtype=object)
```

图 3.8

使用 columns 属性将生成全部列名；类似地，使用 index 属性将生成索引。最后，利用 value 属性将生成 DataFrame 中的数值，并以 NumPy 数组形式予以呈现。

3.2　pandas 操作

pandas 中存在多种操作方法，本节主要考查一些较为常见的操作。

3.2.1　检查数据

当从文件中加载或创建 DataFrame 时，首先需要检查加载后的数据。对此，存在两种相关方法，即 head 和 tail。

其中，head 方法将显示前 5 行数据，如图 3.9 所示。

```
In [12]: data.head()
```

Out[12]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Edu
EmployeeNumber							
1	41	Yes	Travel_Rarely	1102	Sales	1	
2	49	No	Travel_Frequently	279	Research & Development	8	
4	37	Yes	Travel_Rarely	1373	Research & Development	2	
5	33	No	Travel_Frequently	1392	Research & Development	3	
7	27	No	Travel_Rarely	591	Research & Development	2	

5 rows × 34 columns

图 3.9

可以看到，当运行 data.head()方法时，图 3.9 中显示了前 5 行、34 列数据。

当考查最后 5 行数据时，可采用 tail 方法。当运行 data.tail()方法时，图 3.10 显示了最后 5 行数据。通过上述两种方法，可确保数据已被正确地加载。

3.2.2　数据的选取、添加和删除

我们可将 DataFrame 视为一个 Series 字典，其中，每列类似于一个 pandas Series，并采用与字典对象相同的访问方式访问该 Series。相应地，可将 DataFrame 看作是一个索引 Series 对象字典，列的获取、设置和删除操作与字典保持一致。下面考查几个相关示例。

假设需要访问 DataFrame 中的 Age 列，对此，需要编写相应的方法并指明所需访问的列名称，如图 3.11 所示。

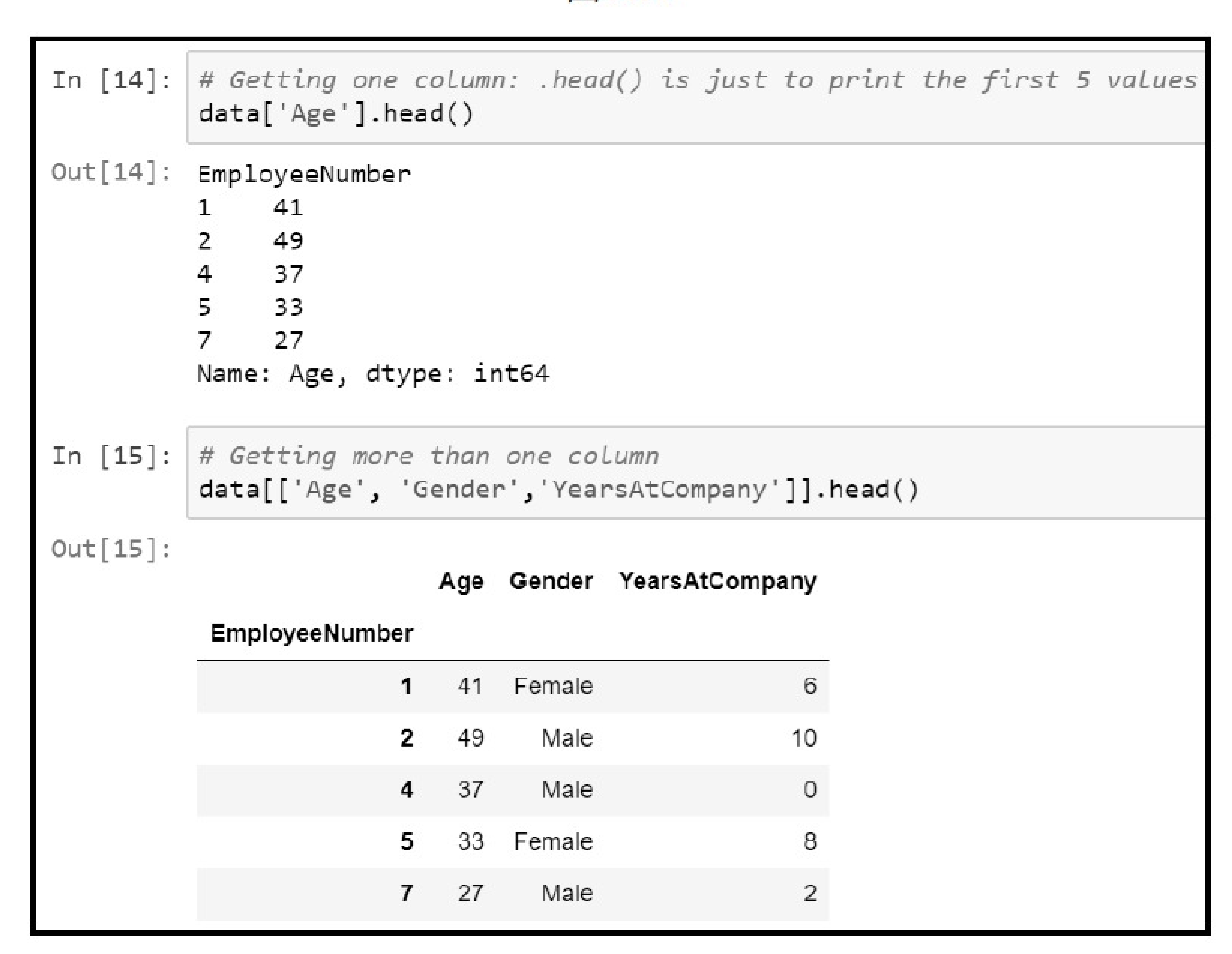
```
In [13]: data.tail()
```

Out[13]:

EmployeeNumber	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Edu
2061	36	No	Travel_Frequently	884	Research & Development	23	
2062	39	No	Travel_Rarely	613	Research & Development	6	
2064	27	No	Travel_Rarely	155	Research & Development	4	
2065	49	No	Travel_Frequently	1023	Sales	2	
2068	34	No	Travel_Rarely	628	Research & Development	8	

5 rows × 34 columns

图 3.10



```
In [14]: # Getting one column: .head() is just to print the first 5 values
         data['Age'].head()

Out[14]: EmployeeNumber
         1    41
         2    49
         4    37
         5    33
         7    27
         Name: Age, dtype: int64

In [15]: # Getting more than one column
         data[['Age', 'Gender','YearsAtCompany']].head()
```

Out[15]:

EmployeeNumber	Age	Gender	YearsAtCompany
1	41	Female	6
2	49	Male	10
4	37	Male	0
5	33	Female	8
7	27	Male	2

图 3.11

其中，当执行 data['Age'].head()和 data[['Age', 'Gender', 'YearsAtCompany']].head()方法时，将从 DataFrame 中所指定的列中获取前 5 行。

注意：

.head()方法用于避免显示 DataFrame 中的所有行，因为这是一个太长的列表。

当向 DataFrame 中添加一列时，可通过 data['AgeInMonths'] = 12*data['Age']方法生成一列，如图 3.12 所示。

```
In [16]: # Adding a column
         data['AgeInMonths'] = 12*data['Age']
         data['AgeInMonths'].head()

Out[16]: EmployeeNumber
         1     492
         2     588
         4     444
         5     396
         7     324
         Name: AgeInMonths, dtype: int64
```

图 3.12

当前，该列并不存在于 DataFrame 中，但通过执行上述方法，新列 AgeInMonths 将被添加至 DataFrame 中。

相应地，一种方式是可采用 del 语句删除 DataFrame 中刚刚创建的列；另一种方式是使用 drop()方法从 DataFrame 中删除列。当采用 drop()方法时，可传递希望删除的列。如果打算删除列，则可指定 axis=1。参数 inplace 则表示就地调整数据对象，同时希望这一修改行为持久化于当前数据结构中。图 3.13 显示了上述各项操作。

```
In [17]: # Deleting a column
         del data['AgeInMonths']
         # the drop method can also be used
         data.drop('EmployeeCount', axis=1, inplace=True)

In [18]: data.columns

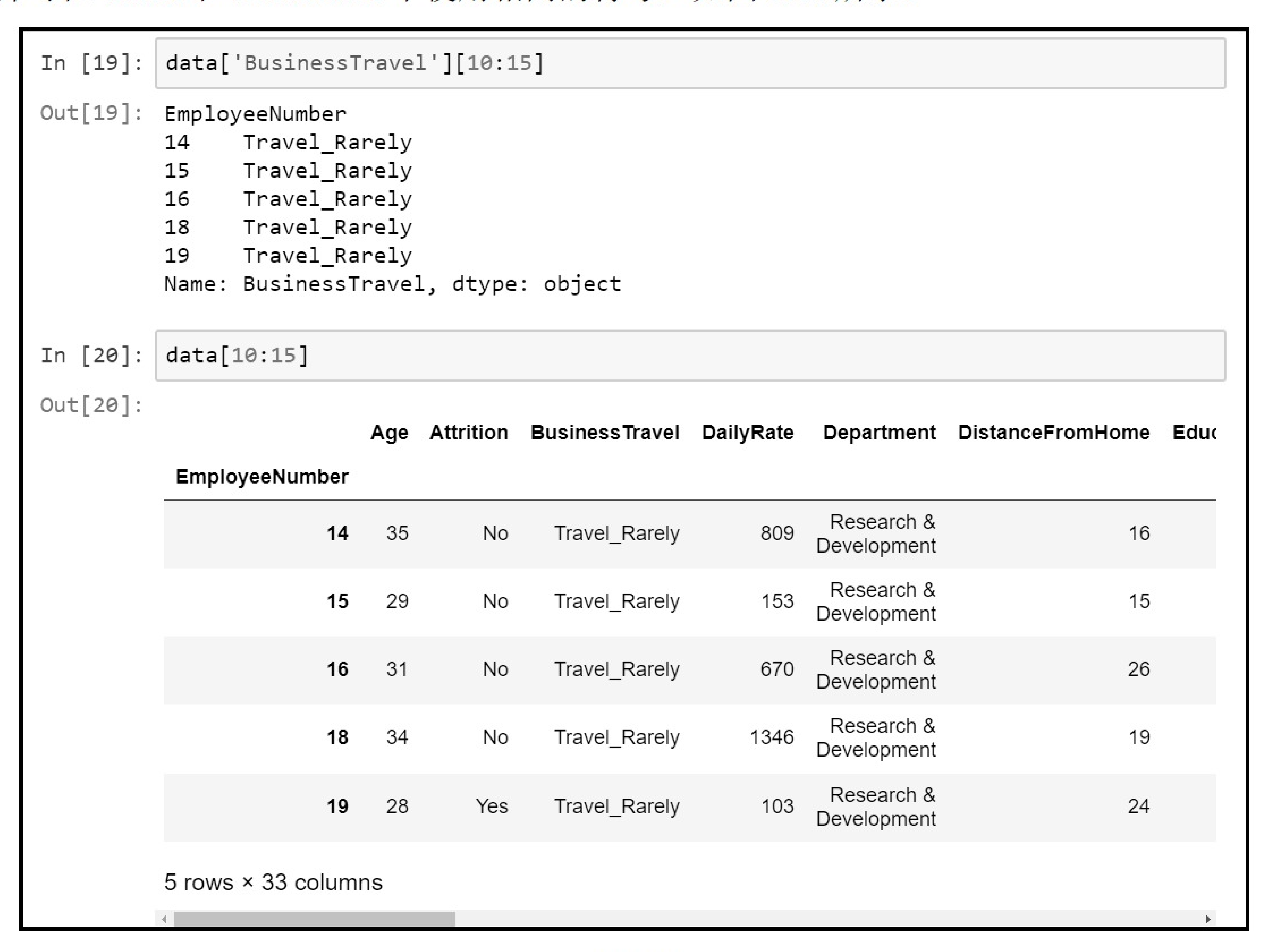
Out[18]: Index(['Age', 'Attrition', 'BusinessTravel', 'DailyRate', 'Department',
                'DistanceFromHome', 'Education', 'EducationField',
                'EnvironmentSatisfaction', 'Gender', 'HourlyRate', 'JobInvolvement',
                'JobLevel', 'JobRole', 'JobSatisfaction', 'MaritalStatus',
                'MonthlyIncome', 'MonthlyRate', 'NumCompaniesWorked', 'Over18',
                'OverTime', 'PercentSalaryHike', 'PerformanceRating',
                'RelationshipSatisfaction', 'StandardHours', 'StockOptionLevel',
                'TotalWorkingYears', 'TrainingTimesLastYear', 'WorkLifeBalance',
                'YearsAtCompany', 'YearsInCurrentRole', 'YearsSinceLastPromotion',
                'YearsWithCurrManager'],
               dtype='object')
```

图 3.13

当再次考查数据列时，可以看到，EmployeeCount 已被删除。

3.2.3　DataFrame 切片

与 NumPy 中的 Series 类似，我们也可以从 pandas Series 和 DataFrame 中获取切片，并可在 Series 和 DataFrame 中使用相同的符号，如图 3.14 所示。



```
In [19]: data['BusinessTravel'][10:15]

Out[19]: EmployeeNumber
         14    Travel_Rarely
         15    Travel_Rarely
         16    Travel_Rarely
         18    Travel_Rarely
         19    Travel_Rarely
         Name: BusinessTravel, dtype: object

In [20]: data[10:15]

Out[20]:
```

EmployeeNumber	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educ
14	35	No	Travel_Rarely	809	Research & Development	16	
15	29	No	Travel_Rarely	153	Research & Development	15	
16	31	No	Travel_Rarely	670	Research & Development	26	
18	34	No	Travel_Rarely	1346	Research & Development	19	
19	28	Yes	Travel_Rarely	103	Research & Development	24	

5 rows × 33 columns

图 3.14

其中，源自位置 10～15（不包含 15）的对应数据被“切取”，并显示于输出结果中。

3.2.4　基于标记的选择操作

此外，我们还可根据标记执行选择操作，这也是为什么索引在数据结构中如此重要的原因。如果希望从某些特定的员工中获取数据，则可使用 loc 方法。首先，通过定义 selected_EmployeeNumbers = [15, 94, 337, 1120]指定需要获取数据的员工。因为在这两个

数据结构中，每个值都与 EmployeeNumber 索引关联，因而可使用该索引访问所需数据，如图 3.15 所示。

```
In [21]: selected_EmployeeNumbers = [15, 94, 337, 1120]

In [22]: data['YearsAtCompany'].loc[selected_EmployeeNumbers]

Out[22]: EmployeeNumber
         15      9
         94      5
         337     2
         1120    7
         Name: YearsAtCompany, dtype: int64
```

图 3.15

图 3.15 中显示了源自 pandas Series 中的数据，其中包含了员工在公司的工作年数。

DataFrame 中也包含了 loc 方法。如果向 DataFrame 中的 loc 方法传递同一列表，将得到与当前标记相关联的全部数据，如图 3.16 所示。

```
In [23]: data.loc[selected_EmployeeNumbers]

Out[23]:
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Edı
EmployeeNumber							
15	29	No	Travel_Rarely	153	Research & Development	15	
94	29	No	Travel_Rarely	1328	Research & Development	2	
337	31	No	Travel_Frequently	1327	Research & Development	3	
1120	29	No	Travel_Rarely	1107	Research & Development	28	

4 rows × 33 columns

```
In [24]: # Getting a single value
         data.loc[94,'YearsAtCompany']

Out[24]: 5
```

图 3.16

如果希望得到 DataFrame 中的特定值或特定单元，则可向 loc 方法中传递两个索引，即行索引和列索引。除此之外，还可利用 iloc 方法并通过位置访问数据。

3.3　数　据　集

假设人力资源总监要求你回答公司员工方面的一些问题，具体如下。

- 数据集中按部门划分有多少名员工？
- 公司员工的总体流失率是多少？
- 公司员工的平均时薪是多少？
- 公司员工的平均工作年限是多少？
- 在公司任职时间最长的 5 名员工是谁？
- 员工整体满意度如何？

3.3.1　数据集中按部门划分的员工数量

当查看数据集中的部门时，可使用 data['Department']语句。随后将得到名为 Department 的列，这表示为 pandas Series；对于每名员工，还将看到该员工所属的部门。因此，当计算 pandas Series 中各部门的员工数量时，可使用 value_counts()方法，如图 3.17 所示。

```
In [25]: data['Department'].value_counts()

Out[25]: Research & Development    961
         Sales                     446
         Human Resources            63
         Name: Department, dtype: int64
```

图 3.17

其中，pandas Series 中 Department 所对应的数值采用表格方式予以显示。

3.3.2　员工的流失率

首先，可使用 data['Attrition']语句查看数据集中的 Attrition 列。在该列中，对应数据表明员工是否仍然在职。对于在职员工，Attrition 等于 No；而离职员工的 Attrition 则等于 Yes。随后，可传递 value_counts()方法以计算每种结果的出现次数。为了得到流失率（离职员工的比率），可使用附加参数 normalize=True。图 3.18 显示了当前操作所对应的代码。

```
In [26]: data['Attrition'].value_counts()

Out[26]: No     1233
         Yes     237
         Name: Attrition, dtype: int64

In [27]: data['Attrition'].value_counts(normalize=True)

Out[27]: No     0.838776
         Yes    0.161224
         Name: Attrition, dtype: float64

In [28]: attrition_rate = data['Attrition'].value_counts(normalize=True)['Yes']
         attrition_rate

Out[28]: 0.16122448979591836
```

图 3.18

当前，为了获得整体流失率，可使用 Yes 索引中的关联标记。因此，在图 3.18 中，我们计算得到了 Yes 索引的数值，即流失率为 16.12%。

3.3.3　平均时薪

pandas Series 中包含了多种统计方法，mean()方法便是较为常用的方法之一，该方法用于计算 pandas Series 的平均值，如图 3.19 所示。

```
In [29]: data['HourlyRate'].mean()

Out[29]: 65.89115646258503
```

图 3.19

通过 mean()方法，可计算得到变量 HourlyRate 的平均值为 65.89。

3.3.4　平均工作年限

为了计算公司内员工的平均工作年限，可采用 mean()方法。除此之外，还存在另一种方法不仅可计算平均值，还可获得 Series 的其他描述性统计结果。具体来说，describe()方法可计算平均值、标准偏差、最小值、最大值和百分比，如图 3.20 所示。

从图 3.20 中可以看到，公司员工的平均工作年限为 7。因此，describe()方法也是一种较为常用的计算方法。

```
In [30]: data['YearsAtCompany'].describe()

Out[30]: count    1470.000000
         mean        7.008163
         std         6.126525
         min         0.000000
         25%         3.000000
         50%         5.000000
         75%         9.000000
         max        40.000000
         Name: YearsAtCompany, dtype: float64
```

图 3.20

3.3.5　任职时间最长的员工

对于 pandas Series 中的数值排序，可使用 sort_values()方法。默认状态下，该方法以升序进行排序。如果不希望采用升序，则可将参数 ascending 设置为 False。鉴于显示结果包含了全部数据列表，因而，可通过 slice 标识仅显示 Series 中的前 5 个元素，如图 3.21 所示。

```
In [31]: data['YearsAtCompany'].sort_values(ascending=False)[:5]

Out[31]: EmployeeNumber
         165     40
         131     37
         374     36
         1578    36
         776     34
         Name: YearsAtCompany, dtype: int64

In [32]: data['YearsAtCompany'].sort_values(ascending=False)[:5].index

Out[32]: Int64Index([165, 131, 374, 1578, 776], dtype='int64', name='EmployeeNumber')
```

图 3.21

其中使用了 index 方法获取员工编号，并以此识别在公司工作年限最长的员工。

3.3.6　员工的整体满意度

数据集中包含了一个 JobSatisfaction 列，用于标识员工的满意度评级，其范围为 1～4。下面使用 head()方法来查看前 5 个观察结果，并创建一个字典，其中数字 1 表示低工

作满意度，值 2 表示中等工作满意度，等等。随后，通过 pandas Series 中的 map()方法将代码转换为相应的类别，这将通过各自的数值关联每个键，也就是说，1 转换为 Low，2 转换为 Medium，等等。接下来，我们将把该 Series 重新分配至使用 map()方法操作的 Series，对应代码如图 3.22 所示。

```
In [33]: data['JobSatisfaction'].head()

Out[33]: EmployeeNumber
         1    4
         2    2
         4    3
         5    3
         7    2
         Name: JobSatisfaction, dtype: int64

In [34]: JobSatisfaction_cat = {
             1: 'Low',
             2: 'Medium',
             3: 'High',
             4: 'Very High'
         }

         Transform this encodings to meaninful labels

In [35]: data['JobSatisfaction'] = data['JobSatisfaction'].map(JobSatisfaction_cat)
         data['JobSatisfaction'].head()

Out[35]: EmployeeNumber
         1    Very High
         2       Medium
         4         High
         5         High
         7       Medium
         Name: JobSatisfaction, dtype: object
```

图 3.22

在图 3.22 中可以看到，我们得到了所需的映射分类。因此，当在 pandas Series 中执行此类转换时，map()方法十分有用。下面将通过 value_counts()方法计算每个类别的出现次数。针对之前提出的问题，获得标准化计数结果可能更加有用，因而可将当前 Series 乘以 100 以获得相应的百分比值，对应代码如图 3.23 所示。

在当前数据集中，31%的员工对自己的工作较为满意（Very High）。此外，我们还可看到其他分类中的百分比数字。

```
In [36]: data['JobSatisfaction'].value_counts()

Out[36]: Very High    459
         High         442
         Low          289
         Medium       280
         Name: JobSatisfaction, dtype: int64

In [37]: 100*data['JobSatisfaction'].value_counts(normalize=True)

Out[37]: Very High    31.224490
         High         30.068027
         Low          19.659864
         Medium       19.047619
         Name: JobSatisfaction, dtype: float64
```

图 3.23

3.4　进一步思考

假设经过第一轮问题之后，人力资源总监还想继续了解员工的一些细节问题，并向你分配了一些新的任务，具体如下。

- ❑ 提交一份员工名单，其中员工的工作满意度较低。
- ❑ 提交一份员工名单，其中员工的工作满意度与工作投入程度均较低。
- ❑ 在以下变量中，通过较低和最高工作满意度对员工进行比较：Age、Department 和 DistanceFromHome。

3.4.1　低满意度员工

为了回答这一问题，可通过布尔 Series 索引一个 Series 或 DataFrame，这称作掩蔽（masking）或布尔索引。对此，首先需要使用比较运算符并利用 data['JobSatisfaction'] == 'Low'值比较 pandas Series，如图 3.24 所示。

在上述代码运行完毕后，针对每名员工，将得到一个包含 True 或 False 的 Boolean Series。其中，True 表示员工的 JobSatisfaction 值等于 Low；False 则表示该值不等于 Low。

我们可以利用该 pandas Boolean Series 索引另一个对象，如 Series 或 DataFrame。当使用 Boolean Series 索引另一个对象时，将得到一个新的 Series 或 DataFrame，其中包含了 True 值的观察结果，如图 3.25 所示。

```
In [42]: data['JobSatisfaction'] == 'Low'

Out[42]: EmployeeNumber
         1        False
         2        False
         4        False
         5        False
         7        False
         8        False
         10        True
         11       False
         12       False
         13       False
         14       False
         15       False
         16       False
         18       False
         19       False
         20        True
         21       False
         22       False
         23       False
         24       False
         26       False
         27        True
         28       False
         30       False
         31        True
```

图 3.24

```
In [43]: data.loc[data['JobSatisfaction'] == 'Low'].index

Out[43]: Int64Index([  10,   20,   27,   31,   33,   38,   51,   52,   54,   68,
                     ...
                     1975, 1980, 1998, 2021, 2023, 2038, 2054, 2055, 2057, 2062],
                    dtype='int64', name='EmployeeNumber', length=289)
```

图 3.25

因此，如果利用 data.loc[data['JobSatisfaction'] == 'Low'].index 索引属性并通过当前 Boolean Series 索引 DataFrame，将会得到较低 JobSatisfaction 值的员工名单。

3.4.2 低工作满意度和低工作参与度的员工

JobInvolvement（工作参与度）列与 JobSatisfaction（工作满意度）列包含了相同的属

性，因而包含了对应的员工号（而非分类）。此处，首先使用之前应用于 JobSatisfaction 列上的 map 转换，如图 3.26 所示。

```
In [44]: JobInvolment_cat = {
             1: 'Low',
             2: 'Medium',
             3: 'High',
             4: 'Very High'
         }
         data['JobInvolvement'] = data['JobInvolvement'].map(JobInvolment_cat)
```

图 3.26

考虑到当前仅考查两列中包含 Low 值的员工，因而可在两个 Boolean Series 上使用 “&” 逻辑运算符，进而得到 JobSatisfaction 和 JobInvolvement 均包含 Low 值的员工列表。再次说明，这里采用了 loc 选取方法以及 index 属性获取特定的需求条件，如图 3.27 所示。

```
In [62]: loc[(data['JobSatisfaction'] == 'Low') & (data['JobInvolvement'] == 'Low')].index

Out[62]: Int64Index([33, 235, 454, 615, 1019, 1037, 1237, 1460, 1478, 1544, 1611, 1622,
                     1905, 1956],
                    dtype='int64', name='EmployeeNumber')
```

图 3.27

图 3.27 显示了 JobSatisfaction 和 JobInvolvement 均包含 Low 值的员工号列表。

3.4.3　员工比较

对于 Low 和 Very High 的 JobSatisfaction，下面将对相应的员工进行比较，并通过 Age、Department、DistanceFromHome 变量对这两个分组进行比较，其间将使用到分组操作。这里，分组操作是指应用一系列的操作行为，包括划分、应用和组合，对应步骤如下。

（1）划分步骤。根据某些标准或目录值，可将 DataFrame 划分为一组 DataFrame。这将生成分组后的对象，其中包含了类似于字典的结构，且每个分组均与不同的键相关联。

（2）针对分组对象使用某个函数进而得到对应结果。

（3）将第（2）步的操作结果分组或整合为一个新的数据结构，该数据结构可以是 new_df 或 new_series。

接下来执行比较操作，并创建一个新的 DataFrame，它只包含作为比较标准条件的那些观察结果，并调用 DataFrame subset_of_interest。此处将采用“|”运算符获取员工的 Series，其中，对应的 JobSatisfaction 为 Low 或 Very High，经适当整合后可使用 shape 属性查看新创建的 DataFrame。随后，可使用操作 value_count 查看最终的观测结果计数情况，如图 3.28 所示。

```
In [46]: subset_of_interest = data.loc[(data['JobSatisfaction'] == "Low") | (data['JobSati
         subset_of_interest.shape

Out[46]: (748, 33)

In [47]: subset_of_interest['JobSatisfaction'].value_counts()

Out[47]: Very High    459
         Low          289


         Name: JobSatisfaction, dtype: int64
```

图 3.28

在上述代码中，可以看到，当前 DataFrame 包含了 748 行和 33 列；另外，具有 Very High 的 JobSatisfaction 显示了 459 个观察结果，而具有 Low 的 JobSatisfaction 则显示了 289 个观察结果。

下面仅考查我们关注的观察结果，并针对新创建的 DataFrame 使用 groupby 操作比较相关变量。另外，还将对从 grouped 操作返回的对象加以命名，对应代码如图 3.29 所示。

```
In [50]: grouped = subset_of_interest.groupby('JobSatisfaction')

In [51]: grouped.groups

Out[51]: {'Low': Int64Index([  10,   20,   27,   31,   33,   38,   51,   52,   54,   68,
                     ...
                     1975, 1980, 1998, 2021, 2023, 2038, 2054, 2055, 2057, 2062],
                    dtype='int64', name='EmployeeNumber', length=289),
          'Medium': Int64Index([], dtype='int64', name='EmployeeNumber'),
          'High': Int64Index([], dtype='int64', name='EmployeeNumber'),
          'Very High': Int64Index([   1,    8,   18,   22,   23,   24,   30,   36,   39,
            40,
                     ...
                     2022, 2024, 2027, 2036, 2040, 2041, 2045, 2052, 2056, 2061],
                    dtype='int64', name='EmployeeNumber', length=459)}
```

图 3.29

不难发现，当对新创建的对象使用 groups()方法后，将得到两个分组，即 Low 分组和 Very High 分组，同时针对每个所关联的分组得到对应的标记或索引。如果希望得到与每个分组所关联的数据，则可使用 get_group()方法，如图 3.30 所示。

```
In [52]: grouped.get_group('Low').head()
Out[52]:
```

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Edu
EmployeeNumber							
10	59	No	Travel_Rarely	1324	Research & Development	3	
20	29	No	Travel_Rarely	1389	Research & Development	21	
27	36	Yes	Travel_Rarely	1218	Sales	9	
31	34	Yes	Travel_Rarely	699	Research & Development	6	
33	32	Yes	Travel_Frequently	1125	Research & Development	16	

5 rows × 33 columns

图 3.30

当执行基于 Low 键的 get_group()方法时，将得到与该 Low 分组所关联的 DataFrame。

注意：

关于上述分组对象，实际上可从原始 DataFrame 中获取 Series。但是，如果尝试通过名称获取 Series，则无法获得 Series，而是得到所谓的 groupby Series。当执行诸如 mean 这一类方法时，将会看到该方法将应用于每个分组上。

因此，如果调用分组对象的 Age Series 并计算均值，值得到每个分组的均值结果。另外，还可使用 groupby Series 中普通 Series 中的各种方法。例如，当使用 describe()方法时，将会看到该方法应用于 Low 分组和 Very High 分组上，如图 3.31 所示。

当对新的 Series 应用相关方法后，Low 分组的平均年龄为 36.9；Very High 分组的平均年龄为 36.7。除此之外，我们还得到了 describe()方法的输出结果，并应用于 Low 分组和 Very High 分组上。从 describe()方法得到的 Series 表示为一个 pandas Series，但该 Series 包含了多重索引。其中，第一级索引表示为 Low 和 Very High 分组；第二级索引表示为统计值的名称，如 mean、std、min 等，如图 3.32 所示。

另外，还可使用 unstack()方法将结果转换为一个 DataFrame。

```
In [51]: grouped['Age']

Out[51]: <pandas.core.groupby.SeriesGroupBy object at 0x000001FBEABD23C8>

In [52]: grouped['Age'].mean()

Out[52]: JobSatisfaction
         Low          36.916955
         Very High    36.795207
         Name: Age, dtype: float64

In [53]: grouped['Age'].describe()

Out[53]: JobSatisfaction
         Low              count    289.000000
                          mean      36.916955
                          std        9.245496
                          min       19.000000
                          25%       30.000000
                          50%       36.000000
                          75%       42.000000
                          max       60.000000
         Very High        count    459.000000
                          mean      36.795207
                          std        9.125609
                          min       18.000000
                          25%       30.000000
                          50%       35.000000
                          75%       43.000000
                          max       60.000000
         Name: Age, dtype: float64
```

图 3.31

```
In [54]: grouped['Age'].describe().unstack()

Out[54]:
```

	count	mean	std	min	25%	50%	75%	max
JobSatisfaction								
Low	289.0	36.916955	9.245496	19.0	30.0	36.0	42.0	60.0
Very High	459.0	36.795207	9.125609	18.0	30.0	35.0	43.0	60.0

图 3.32

接下来，当在不同部门之间进行比较时，可使用 value_counts()和 unstack()方法查看每个分组中每个部门的人员数量，如图 3.33 所示。

比较过程中使用了 normalize 操作，其中，3%的人员（JobSatisfaction 值为 Low）隶属于 Human Resources 部门；66%的人员隶属于 Research & Development 部门；29%的人员隶属于 Sales 部门。类似地，还可得到 Very High 分组的细节信息。

In [55]:
```
grouped['Department'].value_counts().unstack()
```

Out[55]:

Department	Human Resources	Research & Development	Sales
JobSatisfaction			
Low	11	192	86
Very High	17	295	147

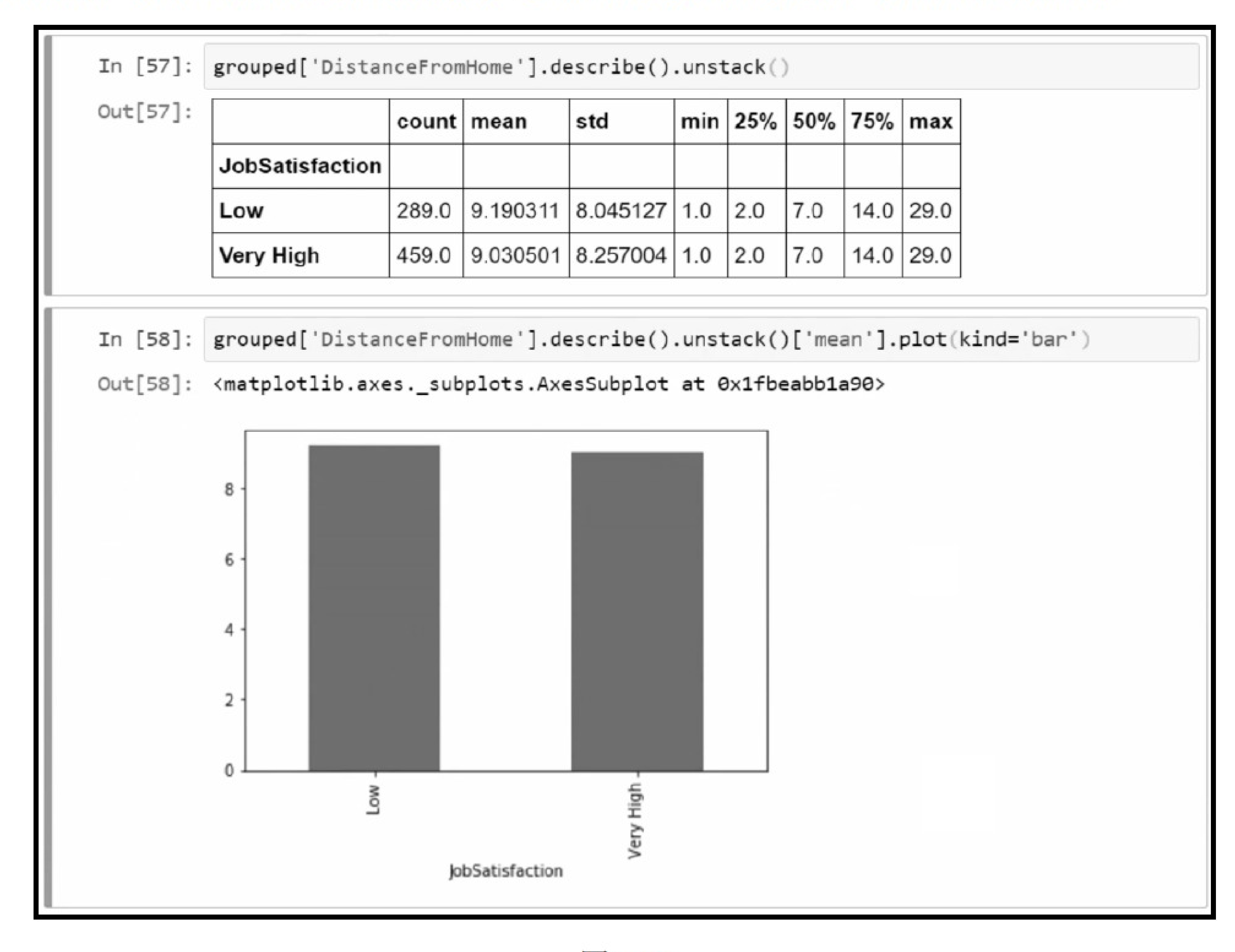
In [56]:
```
100*grouped['Department'].value_counts(normalize=True).unstack()
```

Out[56]:

Department	Human Resources	Research & Development	Sales
JobSatisfaction			
Low	3.806228	66.435986	29.757785
Very High	3.703704	64.270153	32.026144

图 3.33

最后，针对 DistanceFromHome 比较，可使用 describe()和 unstack()方法将其转换为 DataFrame，并在 DistanceFromHome 变量间比较两个分组，如图 3.34 所示。


In [57]:
```
grouped['DistanceFromHome'].describe().unstack()
```

Out[57]:

	count	mean	std	min	25%	50%	75%	max
JobSatisfaction								
Low	289.0	9.190311	8.045127	1.0	2.0	7.0	14.0	29.0
Very High	459.0	9.030501	8.257004	1.0	2.0	7.0	14.0	29.0

In [58]:
```
grouped['DistanceFromHome'].describe().unstack()['mean'].plot(kind='bar')
```

Out[58]: <matplotlib.axes._subplots.AxesSubplot at 0x1fbeabb1a90>

图 3.34

其中，我们得到了源自 DistanceFromHome 比较结果的一个 DataFrame，以及一个源自比较结果平均值的柱状图。

3.5　本 章 小 结

本章介绍了 pandas 及其主要对象，如 Series 和 DataFrame，以及在此基础上的主要操作。此外，我们还通过 pandas 处理了一些每名数据分析师每天都会遇到的问题。

第 4 章将通过一些强大的工具，如 matplotlib 库和 seaborn 库，处理可视化和数据分析问题。

第 4 章　可视化和数据分析

可视化是数据科学和数据分析中的一个重要主题。针对各种功能的可视化操作，Python 提供了多种选择方案。本章将讨论两种较为常见的 Python 可视化库，即 matplotlib 库和 seaborn 库，此外还将讨论与可视化相关的各种 pandas 功能。

本章主要涉及以下主题。

- matplotlib 简介。
- pyplot 简介。
- 面向对象的接口。
- 常见的自定义操作。
- 基于 seaborn 和 pandas 的数据分析。
- 单独分析变量。
- 变量间的关系。

4.1　matplotlib 简介

matplotlib 旨在简化操作，同时实现某些较为复杂的任务。基本上讲，matplotlib 是一个绘制库，可生成多种格式以及交互环境下的高质量图形。本节讨论 matplotlib 的功能、基本概念、图像、子图（坐标系）和坐标轴。除此之外，matplotlib 还可用于 Python 脚本、Python 解释器、Python Shell、Jupyter Notebook、Web 应用程序服务器，以及各种图形用户接口中。

下面考查 Jupyter Notebook，其中包含了较为丰富的 matplotlib 信息。在执行具体操作之间，可首先访问 matplotlib.org，其中包含了相关示例、常见的问题以及图片库。

当与 matplotlib 协同工作时，人们一般会浏览其中的图片库，图 4.1 显示了相应的主界面。

图 4.2 对小提琴图和箱形图进行了比较，同时可查看对应的代码，并可对其进行调整以实现最终的可视化效果。相应地，使用 describe 方法将显示一个较小的 DataFrame，其中包含数据集中每个数值变量的所有描述性统计信息。当前，如果打算逐个可视化所有

这些变量，那么，可以将 hist 方法应用于 DataFrame，而不仅仅是一个 Series，这也是 pandas 和可视化功能的一个优点。



图 4.1

图 4.2 中可以看到部分代码内容。读者可访问 matplotlib 官方站点（matplotlib.org），以查看完整的代码内容。

在讨论 matplotlib 库的主要概念之前，下面首先介绍 matplotlib 中的一些基本术语，如绘图窗口、子图/坐标系、坐标轴，如图 4.3 所示。

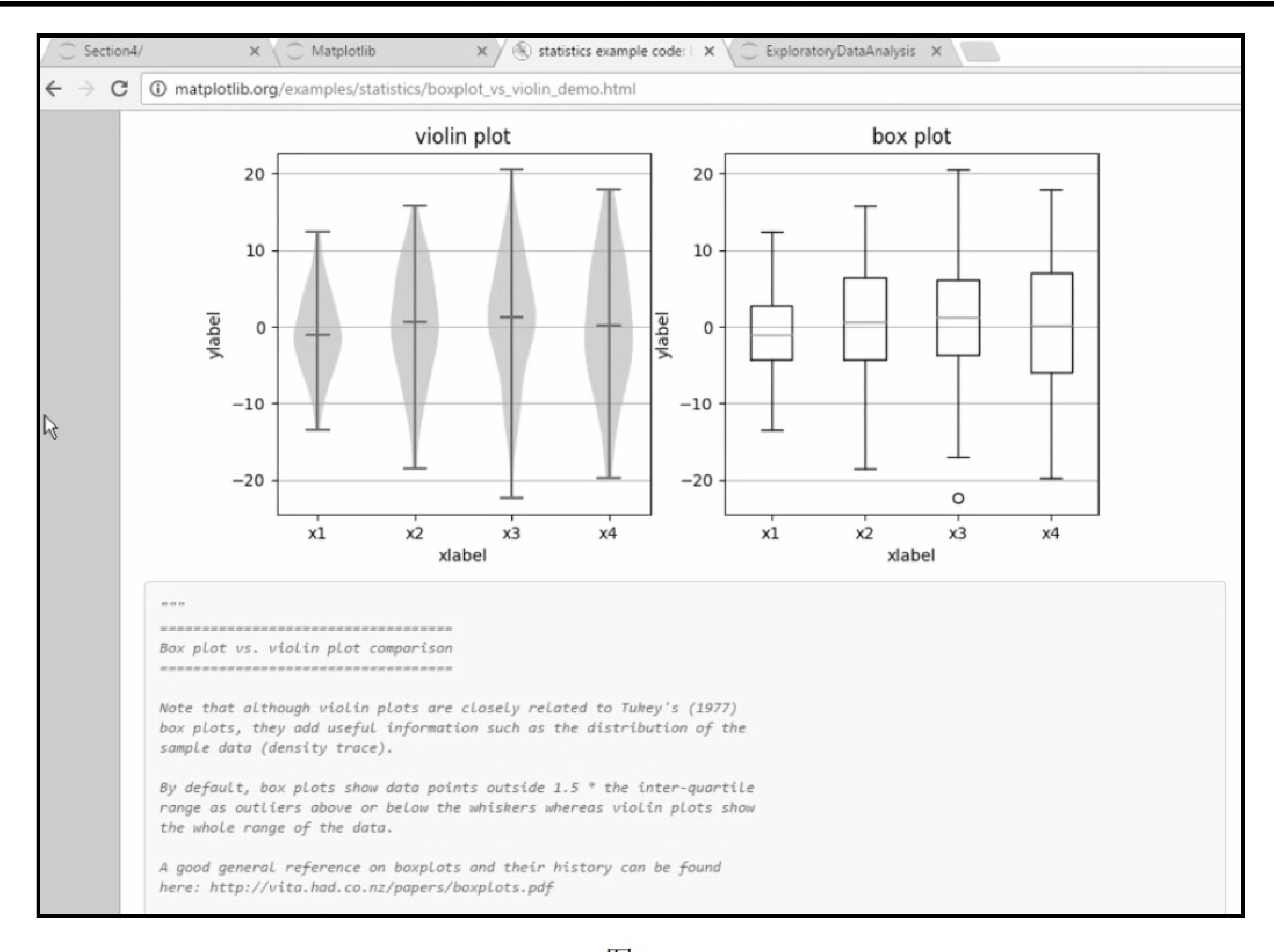

Section4/
Matplotlib
statistics example code:
ExploratoryDataAnalysis
matplotlib.org/examples/statistics/boxplot_vs_violin_demo.html
violin plot
box plot
ylabel
xlabel
x1
x2
x3
x4
"""
==================================
Box plot vs. violin plot comparison
==================================
Note that although violin plots are closely related to Tukey's (1977)
box plots, they add useful information such as the distribution of the
sample data (density trace).
By default, box plots show data points outside 1.5 * the inter-quartile
range as outliers above or below the whiskers whereas violin plots show
the whole range of the data.
A good general reference on boxplots and their history can be found
here: http://vita.had.co.nz/papers/boxplots.pdf

图 4.2

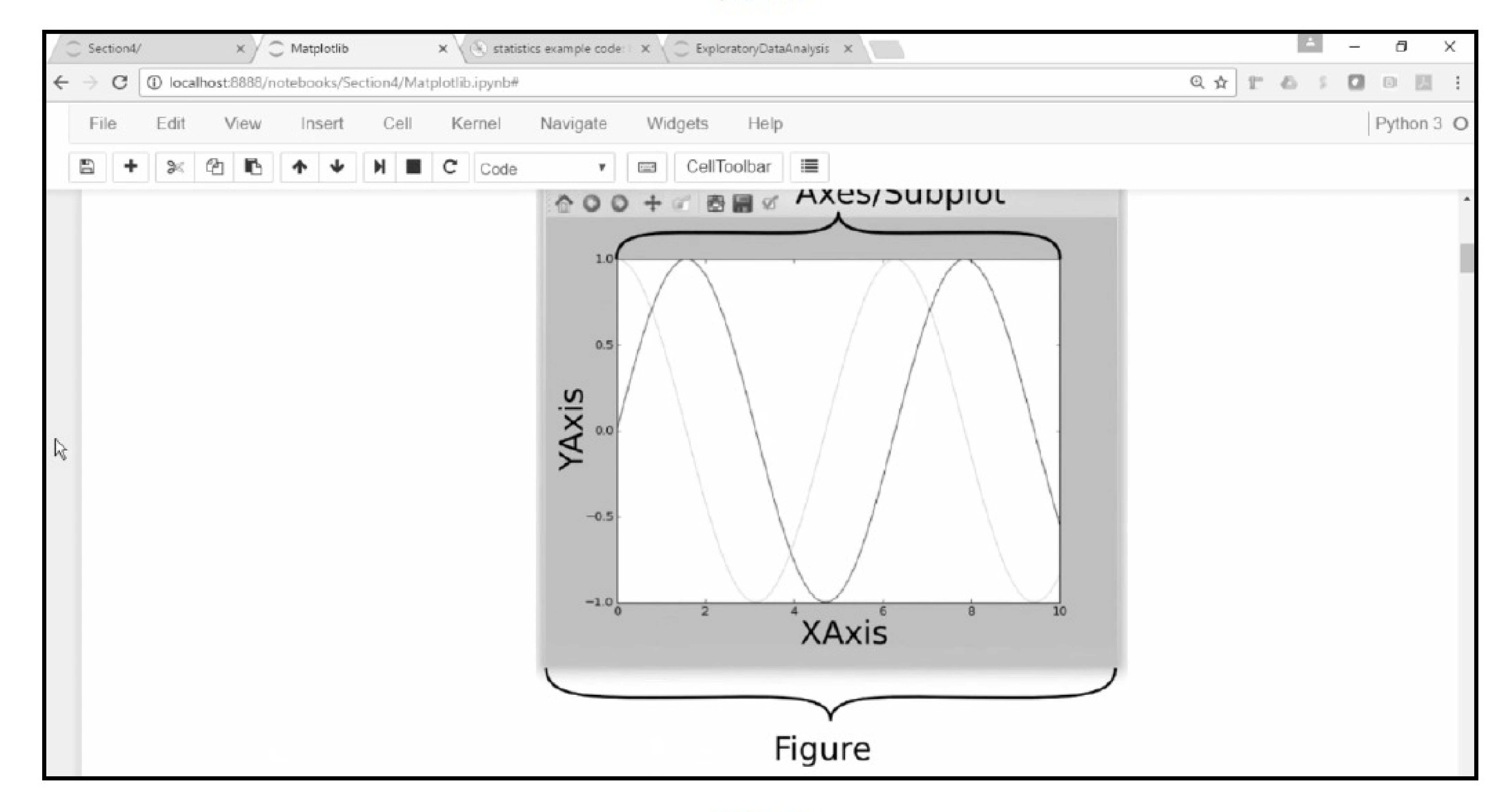

Section4/
Matplotlib
statistics example code:
ExploratoryDataAnalysis
localhost:8888/notebooks/Section4/Matplotlib.ipynb#
File
Edit
View
Insert
Cell
Kernel
Navigate
Widgets
Help
Python 3
Code
CellToolbar
Axes/Subplot
YAxis
XAxis
Figure

图 4.3

下面对图 4.3 中的各项内容进行逐一考查。

- 绘图窗口。绘图窗口表示层次结构中的第一层容器，同时也是包含绘制内容的整体窗口。我们可设置多个绘制窗口，且绘制窗口中可包含多个坐标系。
- 坐标系/子图。大部分绘制内容与某个轴向或子图相关，且涉及多个组件，如 x 轴和 y 轴。此外，图 4.3 中还包含了绘制区域、刻度线等。作为子图的一部分内容，其中还包含了其他对象，如 x 轴及 x 轴中的 x 标记、x 刻度，以及该刻度的标记。这也是 matplotlib 中基本的层次结构。
- 坐标轴。层次结构最上方包含了绘图窗口，该窗口内包含了多个子图。图 4.3 中仅包含了一个子图。相应地，绘图窗口中还可包含多个子图。每个子图还包含了其他元素，一般为 x 轴、y 轴和其他元素。

4.2　pyplot 简介

下面将通过 pyplot 接口使用 matplotlib，这一过程涉及 pyplot 接口和相关示例。

在 Jupyter Notebook 中，应留意以下命令。

```
%matplotlib inline
```

这也是通知 Jupyter Notebook 的一种基本操作方式，进而查看 Jupyter Notebook 中的绘图。若当前绘制窗口中未使用该命令并执行上述代码行，绘制结果将显示于不同的窗口中。

基本上讲，pyplot 表示为一个命令样式函数集合，并使得 matplotlib 的工作方式与 MATLAB 类似，其工作理念可描述为：持有一组函数，且每个函数均可对绘制窗口进行修改，该绘制窗口被视为当前的绘制窗口。因此，每个函数均可对绘制窗口执行某种操作。例如，可生成一个绘制窗口；可在某个绘制窗口中创建一个绘制区域；可在绘制窗口的子图中绘制一条直线；可修改标记；等等。当使用 pyplot 时，需要知晓哪一个窗口是当前绘制窗口，考查以下示例。

（1）首先需要向当前会话中导入 matplotlib，如下所示。

```
import matplotlib as plt
```

（2）第一条命令是设置 plt 模块和 pyplot 模块中的 plot 函数，并传递一个数值列表。因此，执行该命令后将生成一个绘制窗口，如图 4.4 所示（尽管图 4.4 中无法看到该绘制窗口）。其中，绘制窗口中包含了一个子图，该子图绘制了一条直线，即列表中数值的图形表现方式。

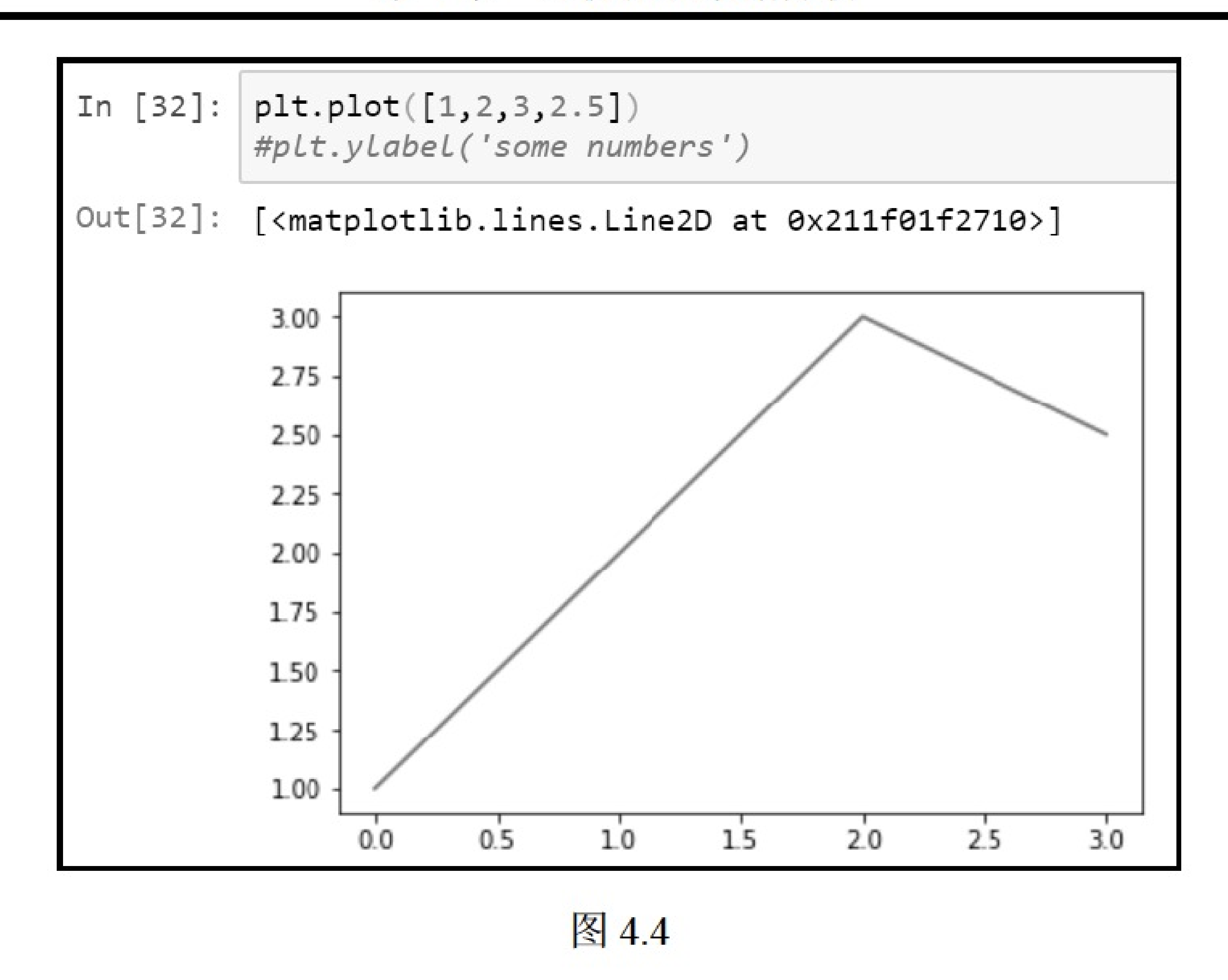


图 4.4

在图 4.4 中，可以看到该函数的操作内容，且该窗口被视为当前的绘制窗口。对于其他函数，例如调用 plt.ylabel，将在 y 轴上设置一个标记，即 ylabel。在当前示例中，该标记被视为一些数字。当再次运行时，对应结果如图 4.5 所示。其中，对应标记显示于 y 轴上。

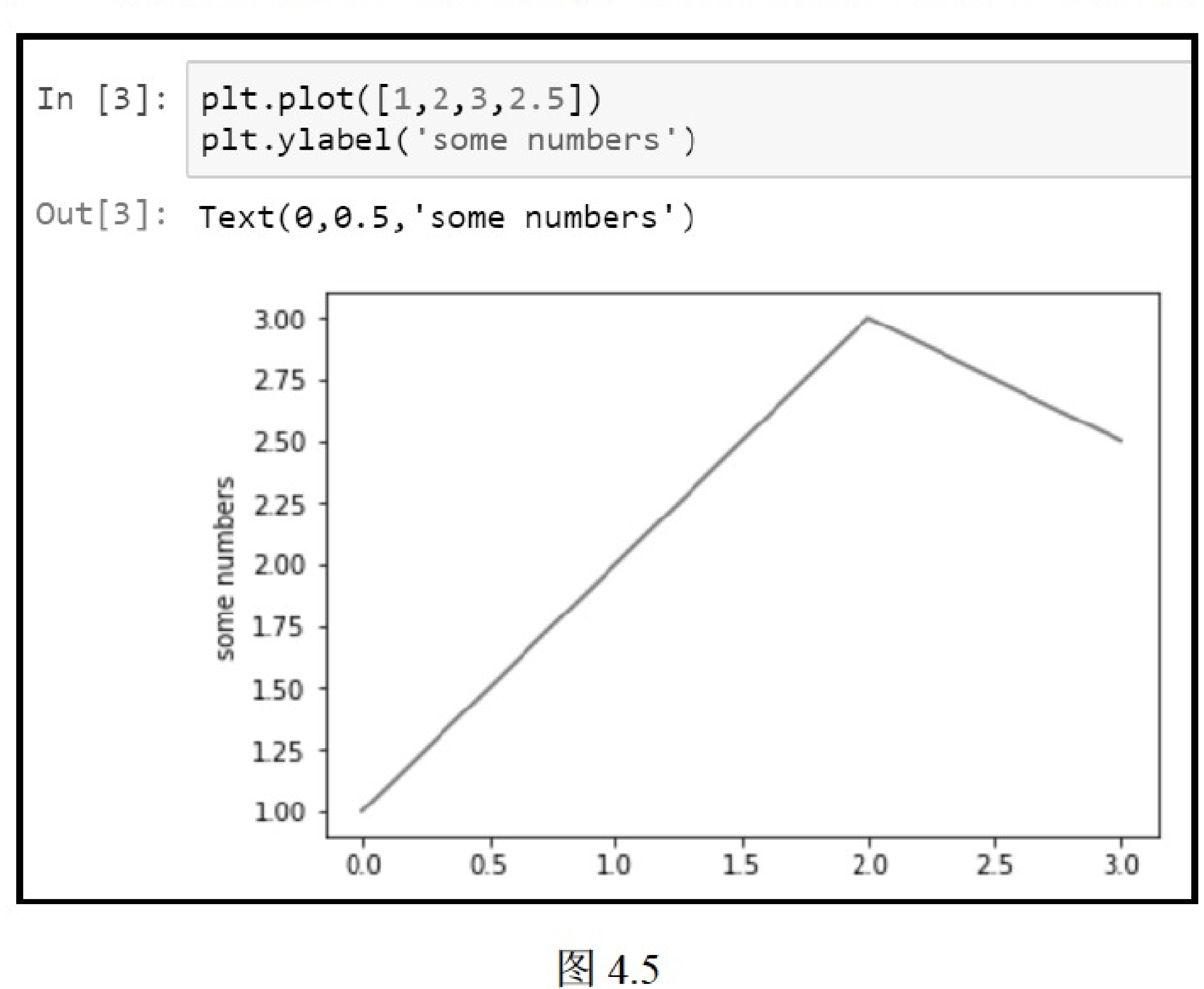


图 4.5

（3）pyplot 中较为常用的函数是 plot 函数，该函数可接收多个参数。例如，若传递两个数值列表，且假设第一个列表为 x 坐标，第二个列表为 y 坐标。在当前示例中，默认状态下，该函数将绘制一条直线，如图 4.6 所示。

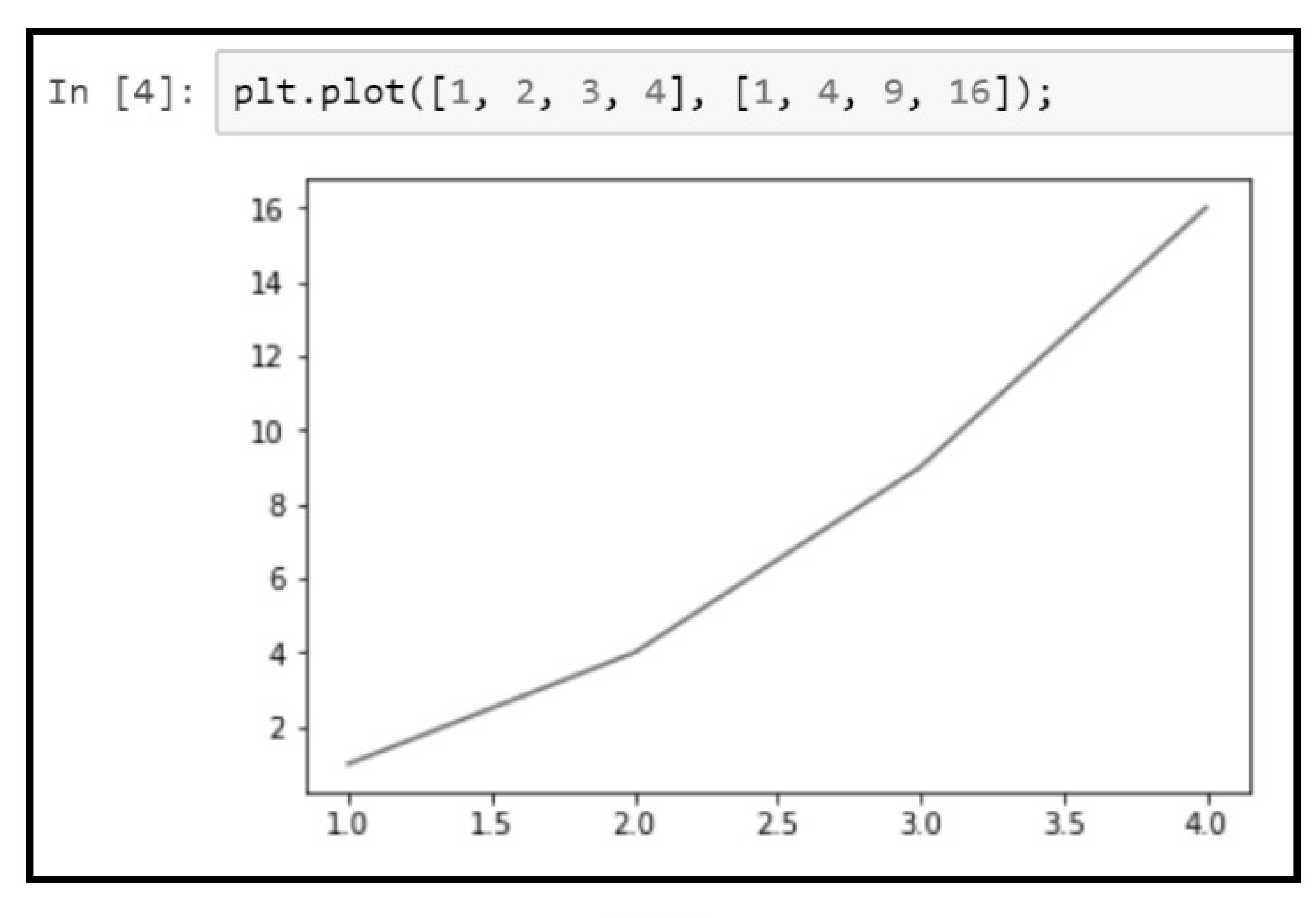


图 4.6

上述界线的工作方式可描述为：通过调用函数序列构建图，并应用于当前绘制窗口和当前子窗口中。下面考查如何通过 pyplot 构建图，具体步骤如下。

（1）向 plot 函数添加两个列表，如前所述，默认行为是绘制一条直线。因此，可以看到所显示的 x 和 y 值、直线的颜色（lightblue），且直线的尺寸（linewidth）为 3，对应的输出结果如图 4.7 所示。

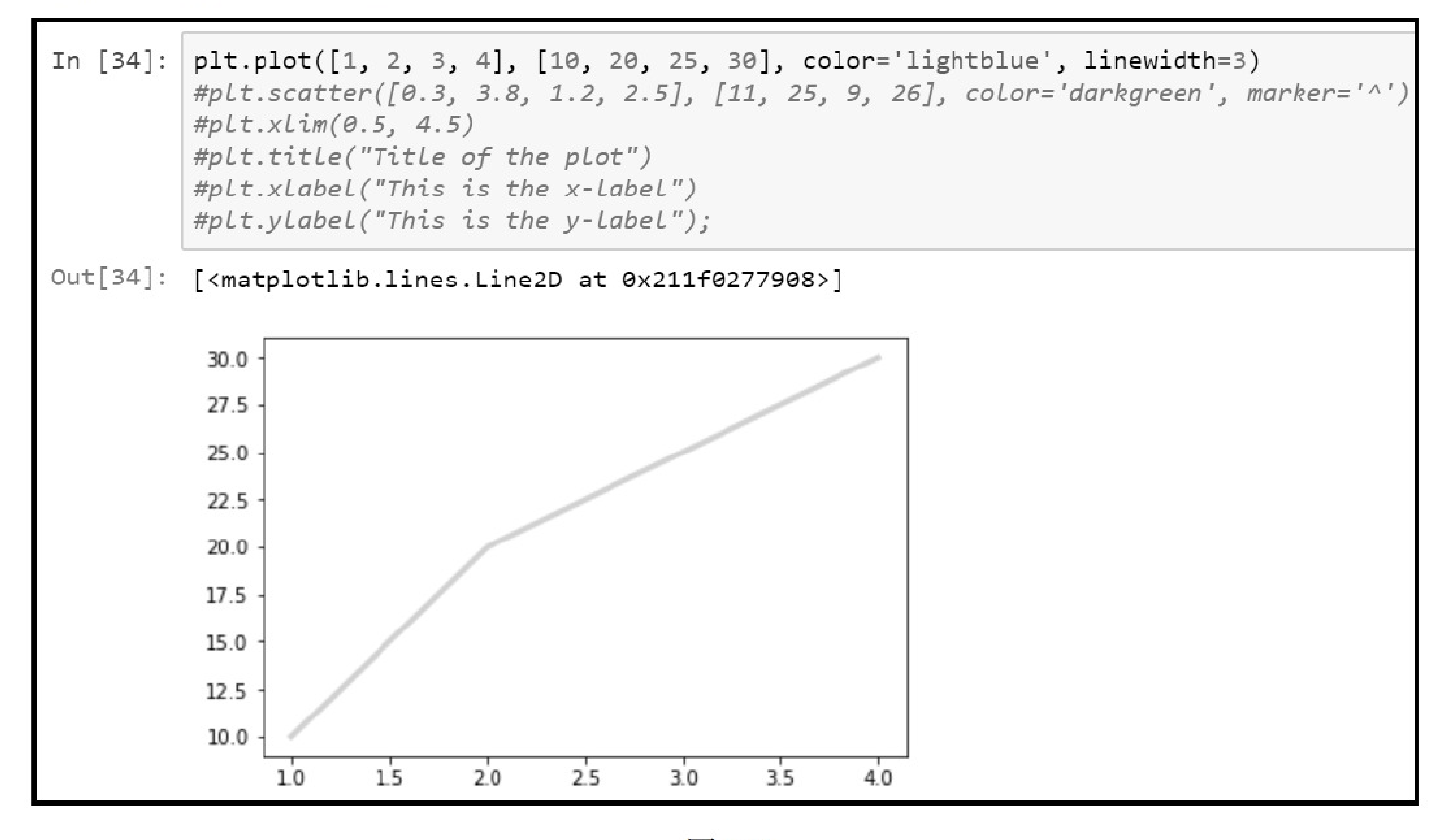


图 4.7

从图 4.7 中可以看到，一些函数已被注释掉。下面将尝试逐行移除注释，并查看每个函数带来的变化内容。

（2）散点图对所提供的坐标进行绘制，而非绘制一条直线。在当前示例中，对应坐标分别为 x 坐标和 y 坐标。除此之外，图中还显示了一些绘制标记。对此，可定义一个参数并将标记的颜色设置为 darkgreen，对应结果如图 4.8 所示。

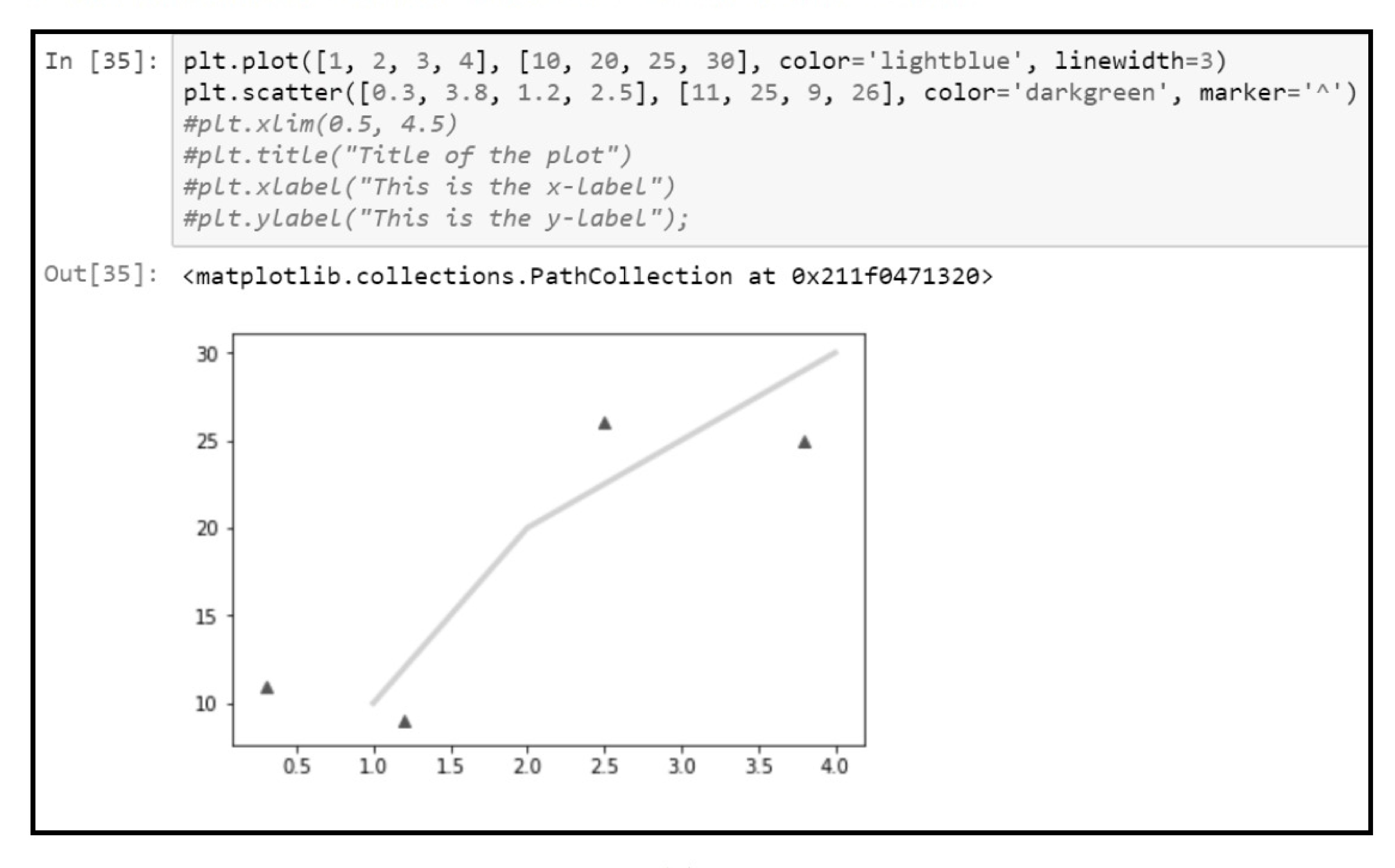


图 4.8

（3）下面将通过另一个函数修改 x 的极限值。其中，最小值为 0.5，最大值为 4.5。也就是说，x 轴上的绘制范围为 0.5～4.5。在图 4.9 中可以看到调整后的效果，同时逐行去除了注释内容。

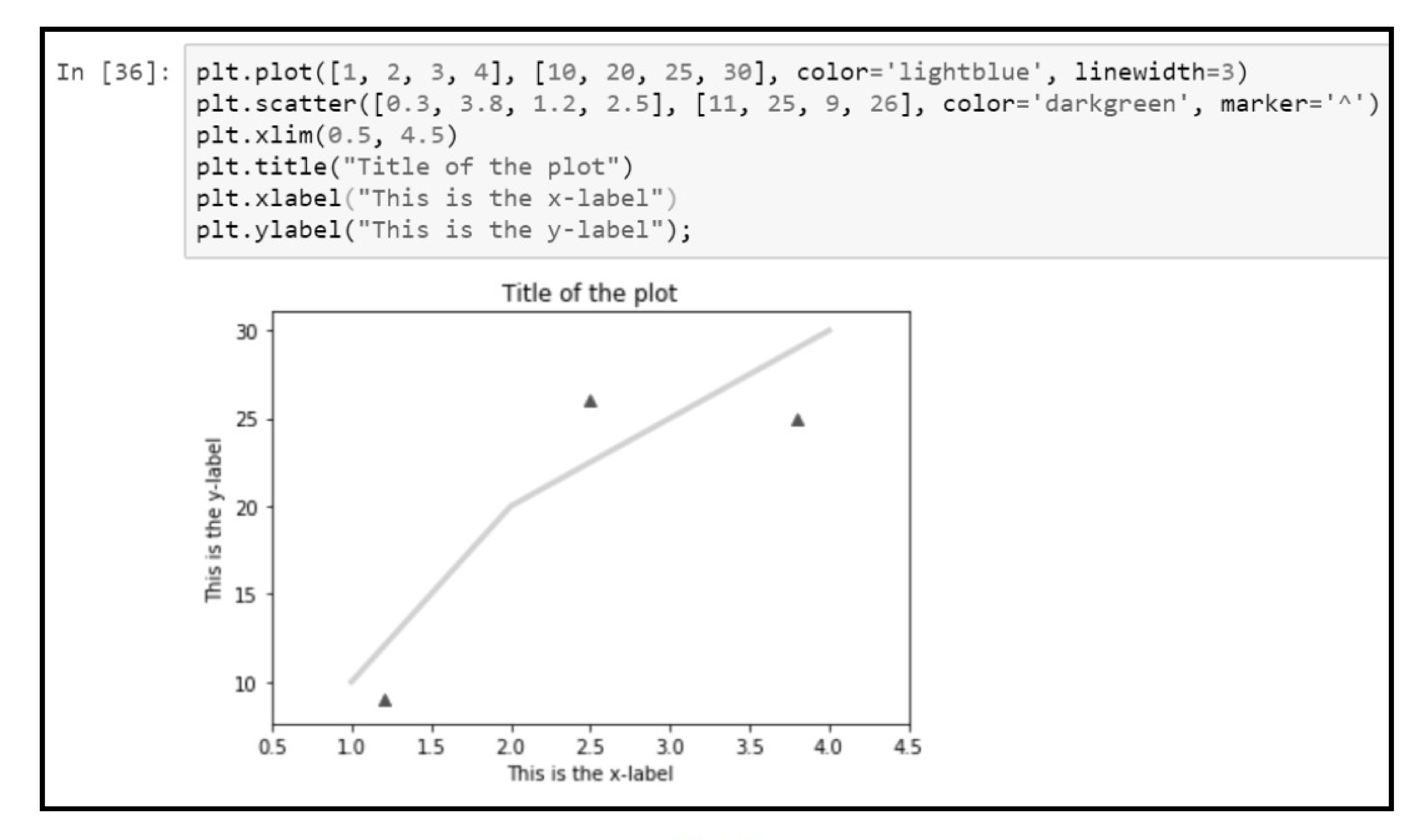
（4）下一个函数将添加一个绘制标题并调整绘制标记。在图 4.10 中，可以看到添加了对应的标题、x 标记和 y 标记。

当与子图或绘制窗口协同工作时，一种较为方便的方法是使用 pyplot 接口。但在单一绘制窗口中与多个绘制窗口或多个子图协同工作时，这一过程往往会产生混淆。

```
In [35]: plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)
         plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')
         plt.xlim(0.5, 4.5)
         #plt.title("Title of the plot")
         #plt.xlabel("This is the x-label")
         #plt.ylabel("This is the y-label");

Out[35]: <matplotlib.collections.PathCollection at 0x211f0471320>
```

图 4.9



```
In [36]: plt.plot([1, 2, 3, 4], [10, 20, 25, 30], color='lightblue', linewidth=3)
         plt.scatter([0.3, 3.8, 1.2, 2.5], [11, 25, 9, 26], color='darkgreen', marker='^')
         plt.xlim(0.5, 4.5)
         plt.title("Title of the plot")
         plt.xlabel("This is the x-label")
         plt.ylabel("This is the y-label");
```


图 4.10

（5）运行当前示例，相关内容如图 4.11 所示。

```
In [6]: import numpy as np
        def f(t):
            return np.exp(-t) * np.cos(2*np.pi*t)

        t1 = np.arange(0.0, 5.0, 0.1)
        t2 = np.arange(0.0, 5.0, 0.02)
```

图 4.11

（6）根据图 4.11 中的数据，可创建一个包含两个子图的绘制窗口。相应地，首先将生成一个绘制窗口，并于随后添加一个子图。此处需要构建一个两行一列的子图网格。这里将绘制所生成并定义的 t1、f(t)，对应结果如图 4.12 所示。

```
In [6]: import numpy as np
        def f(t):
            return np.exp(-t) * np.cos(2*np.pi*t)

        t1 = np.arange(0.0, 5.0, 0.1)
        t2 = np.arange(0.0, 5.0, 0.02)

In [7]: plt.figure()
        plt.subplot(2, 1, 1)
        plt.plot(t1, f(t1), 'bo')

        plt.subplot(2, 1, 2)
        plt.plot(t2, np.cos(2*np.pi*t2), 'r--')

        plt.ylabel("Y label"); # Which subplot is modifying this function?
```

图 4.12

（7）图 4.12 中使用了另一个命令调整绘制过程中的参数，且仍将得到两行一列的网格，即第二幅绘图，其中将绘制 t2 对象并使用了标记。为了对标记进行修改，需要保持当前子图，如图 4.13 所示。

因此，由于可以确定所引用的对象，因而建议选用面向对象的接口。在当前示例中，当使用某个函数时，应确保知晓对应的子图或当前绘制窗口。

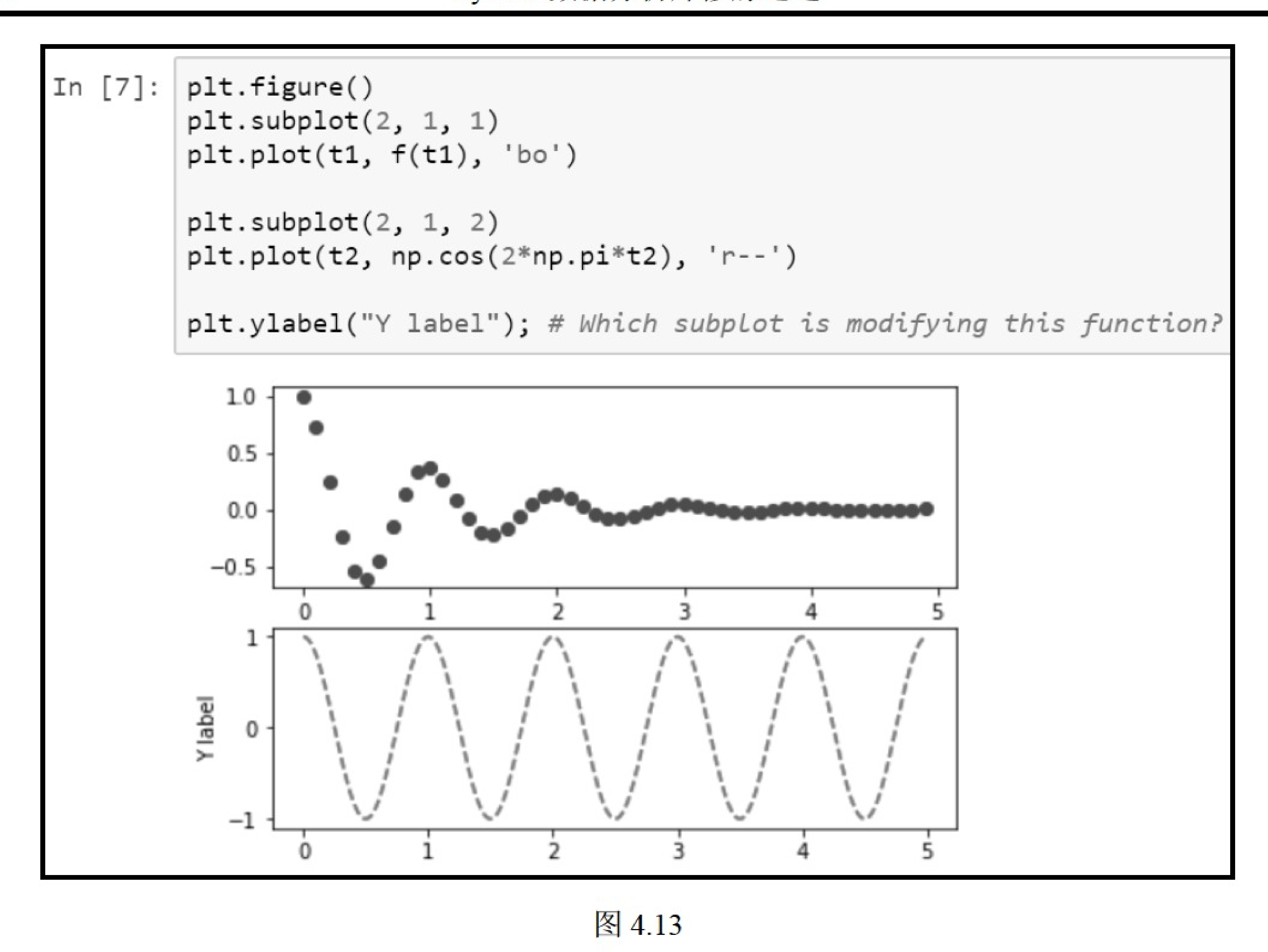


图 4.13

4.3　面向对象接口

本节讨论面向对象接口 matplotlib，其中主要涉及以下主题。

- 面向对象接口。
- 利用子图网格创建绘制窗口。
- 通过相关方法进行绘制。

在面向对象接口中，需要生成对象并针对每个对象调用相应的方法，进而面向该对象进行调整。下面考查一个简单的示例，并利用面向对象接口生成绘图。对此，一般可采用 plt.subplots 同时创建两个对象。当创建绘制窗口对象时，一般将其称作 fig，而另一个轴对象则称作 ax。在当前示例中，默认函数将生成包含单一轴对象的绘制窗口，或者是一个子图。由于我们需要创建自己的对象，因而须进行适当调整。当调整绘图、标题或设置标记时，可使用此类对象上的相关方法。下面考查一些示例，并将相应的示意图作为参考点。其间将使用对象上的 plot()方法，绘制 NumPy 数组的 x 和 y1 数组。接下来

使用相同的对象，并绘制 x 和 y2。随后，针对同一对象，将分别设置 x 标记、y 标记以及标题，对应结果如图 4.14 所示。

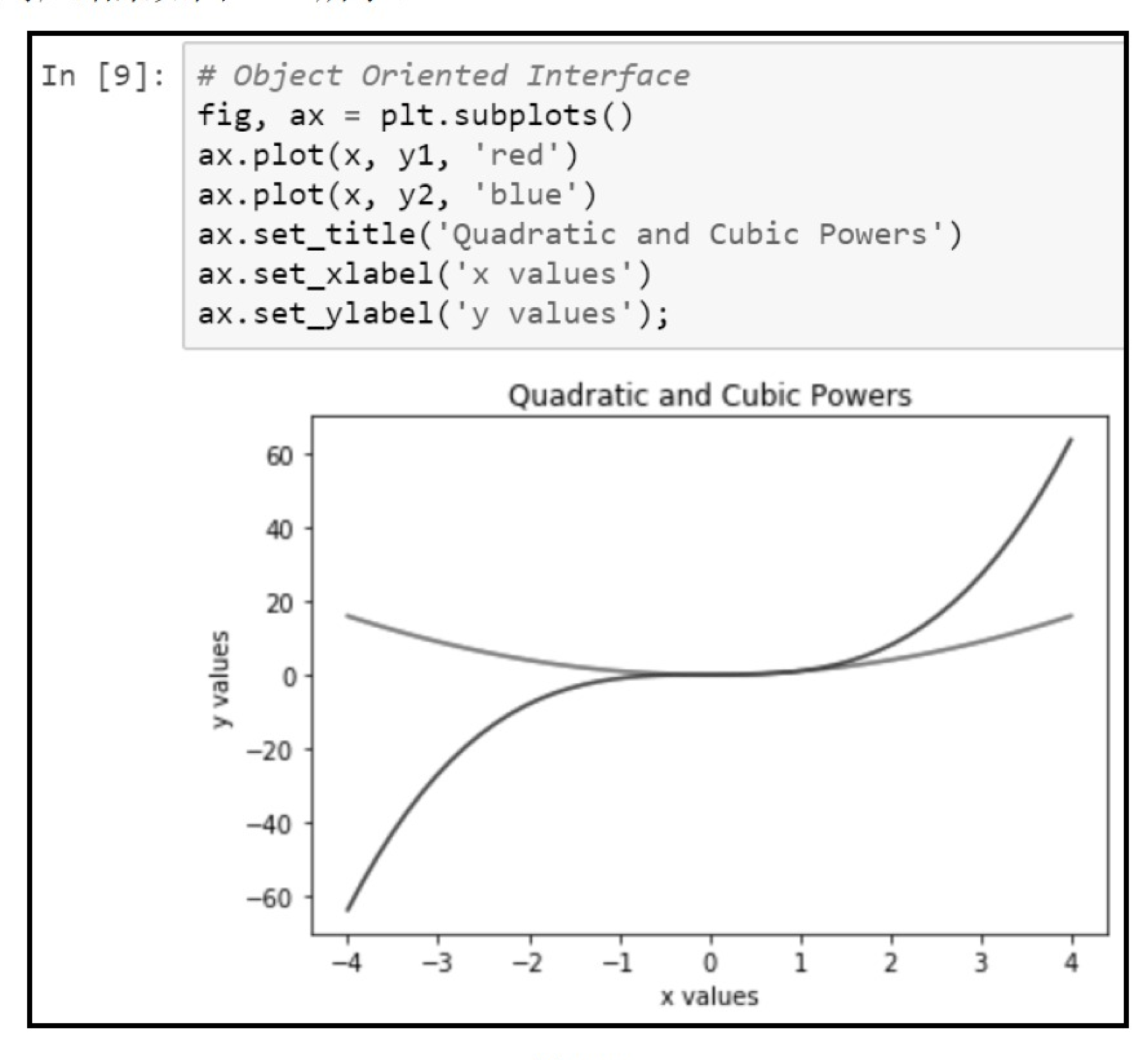


图 4.14

下面考查如何利用 pyplot 接口生成相同的绘制结果。在 pyplot 接口中，无须使用显式的命令创建绘制窗口，plot 函数将执行此项任务。另外，还可在此类函数中传递多个参数。首先，需要传递前两个坐标对：第一个坐标对，即 x 和 y1 坐标，以表明绘制一条红线；然后可传递第二个坐标对，即 x 和 y2 坐标，以表明绘制一条蓝线。接下来，可针对该对象设置 x 标记和 y 标记，对应的输出结果如图 4.15 所示。

鉴于重新生成相同的绘制结果，因而代码量也有所减少。这里的问题是，面向对象接口的优势是什么？当与某个绘制窗口中的多个子图协同工作时，面向对象接口的优点即可显现出来。因此，如果在一个绘制窗口中仅生成一个子图，那么，较好的方法是使用面向对象接口 pyplot。当使用多个子图时，面向对象的优点将更加突出。

因此，为了生成子图网格，可使用 plt.subplots 函数，并可指定所需的行数和列数。

在图 4.16 中显示了 4 个子图。

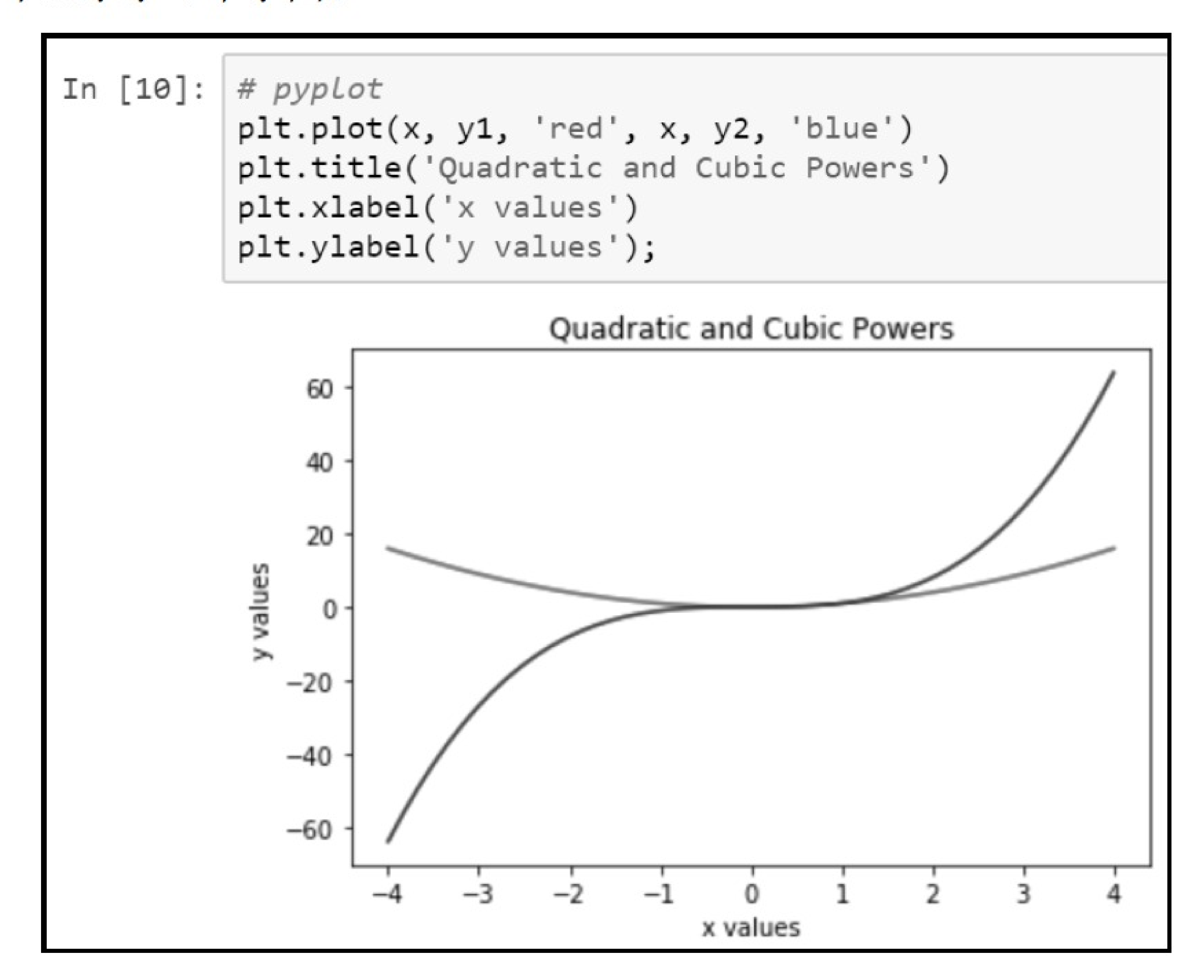


图 4.15

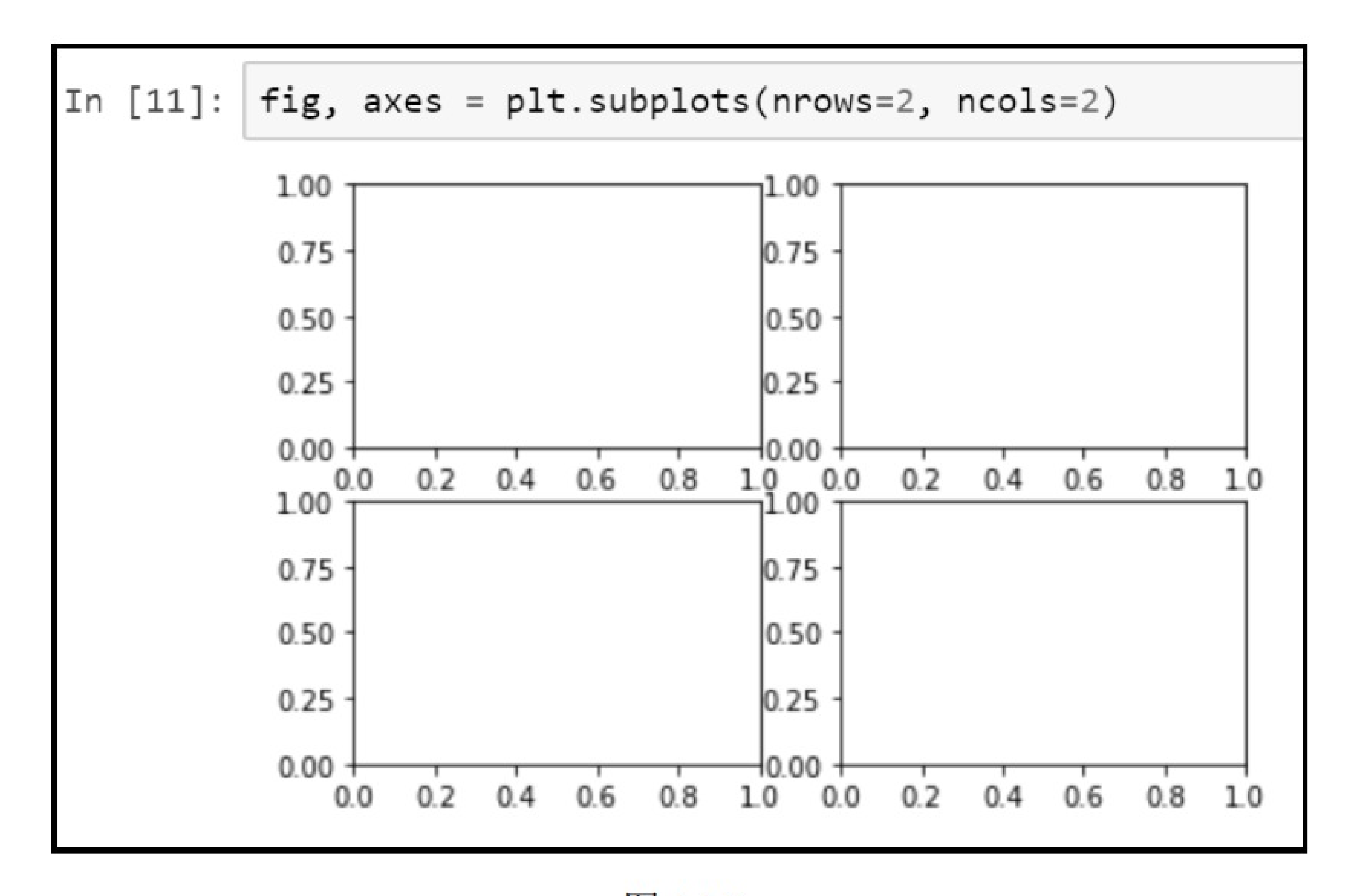
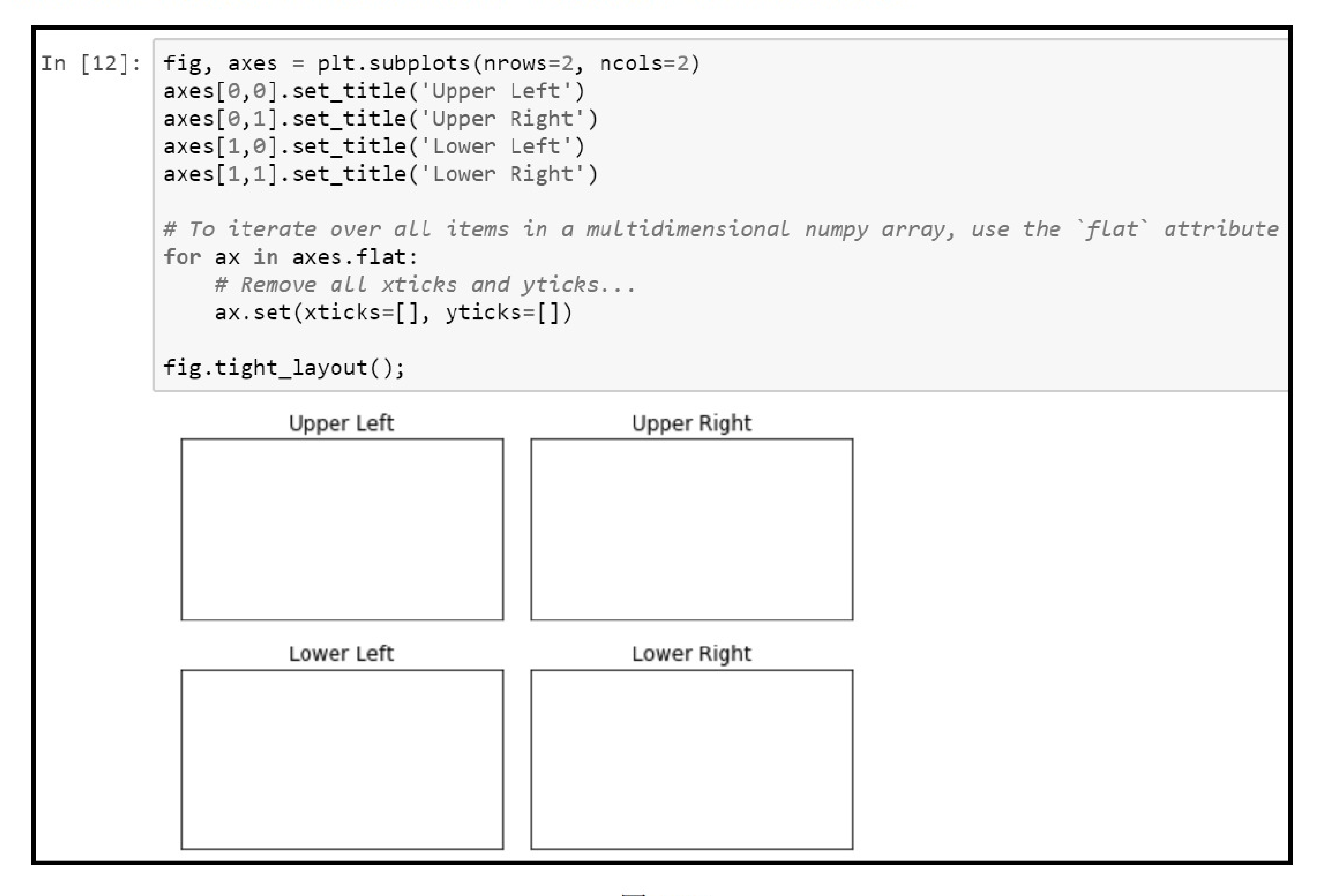


图 4.16

当前，可对每个对象加以引用，这也是面向对象接口的优点之一。下面考查另一个示例，并再次运行代码创建两个对象，即绘制窗口对象和坐标轴对象。在当前示例中，坐标轴表示为包含 4 个子图的多维 NumPy 数组。当访问子图时，需要使用 NumPy 数组符号。因此，第一个子图表示为索引（0,0）的子图。如果需要对该对象进行适当调整，则可使用相应的方法完成这一操作。在当前示例中，假设需要修改标题内容，并设置该对象的标题。也就是说，在索引(0,0)中，可将标题设置为 Upper Left。通过对应的索引，可引用每个子图，并利用具体的操作方法。此处将针对每个对象调整标题内容。考虑到对象 axes 表示为一个 NumPy 数组，因而可以迭代是否要对每个对象、每个子图进行类似的更改。

因此，可针对每个子图使用 flat 属性，并将其称作 ax。此处，将 xticks 和 yticks 设置为空表，并对其进行移除。当运行代码时，可以看到，xticks 或 yticks 将不复存在。由于这里设置了标题，因而此处显示了全部标题，如图 4.17 所示。


```
In [12]: fig, axes = plt.subplots(nrows=2, ncols=2)
         axes[0,0].set_title('Upper Left')
         axes[0,1].set_title('Upper Right')
         axes[1,0].set_title('Lower Left')
         axes[1,1].set_title('Lower Right')

         # To iterate over all items in a multidimensional numpy array, use the `flat` attribute
         for ax in axes.flat:
             # Remove all xticks and yticks...
             ax.set(xticks=[], yticks=[])

         fig.tight_layout();
```


图 4.17

由于创建了绘制窗口对象，因而可使用 tight_layout 这一类绘制窗口方法，且仅关注

子图的呈现方式，以避免彼此间处于重叠状态。基本上讲，这可视为面向对象接口背后所蕴含的理念。我们创建了相关对象，并于随后针对所创建的每个对象应用相关方法。在图 4.18 中包含了多个实例以及生成结果。

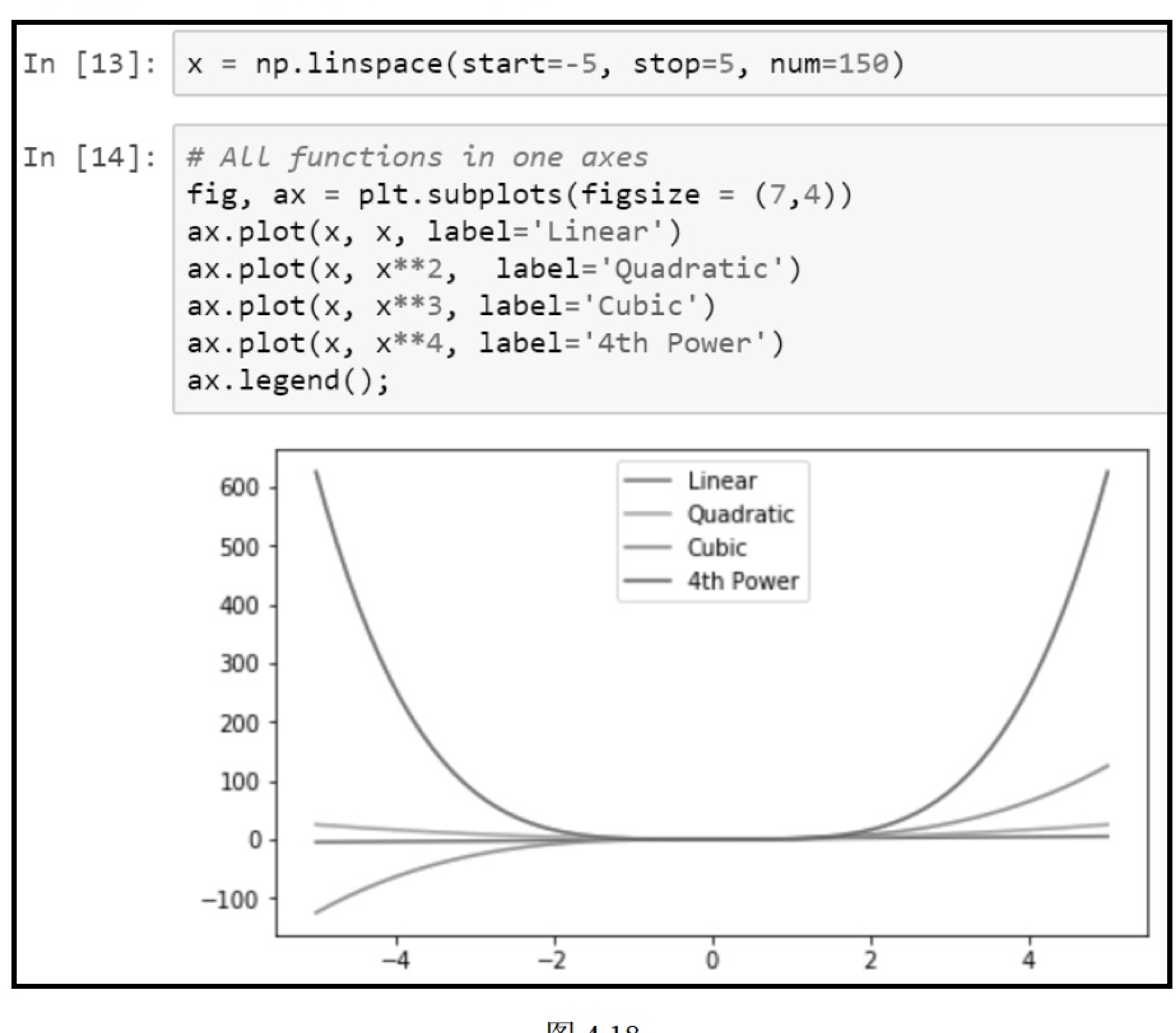


图 4.18

下面生成包含单一子图的独立绘制窗口，并绘制 x 的 4 次方，同时查看如何在不同的子图中进行绘制。在单元中，将创建包含两行两列网格的子图，因而总计 4 个子图。接下来使用面向对象接口针对第一个子图设置标题，并针对 Linear 函数设置绘图。对于第二个绘图，其索引为(0,1)，(0,1)，将设置标题并绘制 x 和 x 的平方，对应结果如图 4.19 所示。

在图 4.19 中可以看到，其中包含了 4 个子图，且每个子图对应一个函数。在图 4.20 中，还可进一步查看 x 的 10 次方。因此，这体现了面向对象接口的工作方式，同时也是 matplotlib 的应用方式。

因此，面向对象接口的方便性可见一斑。

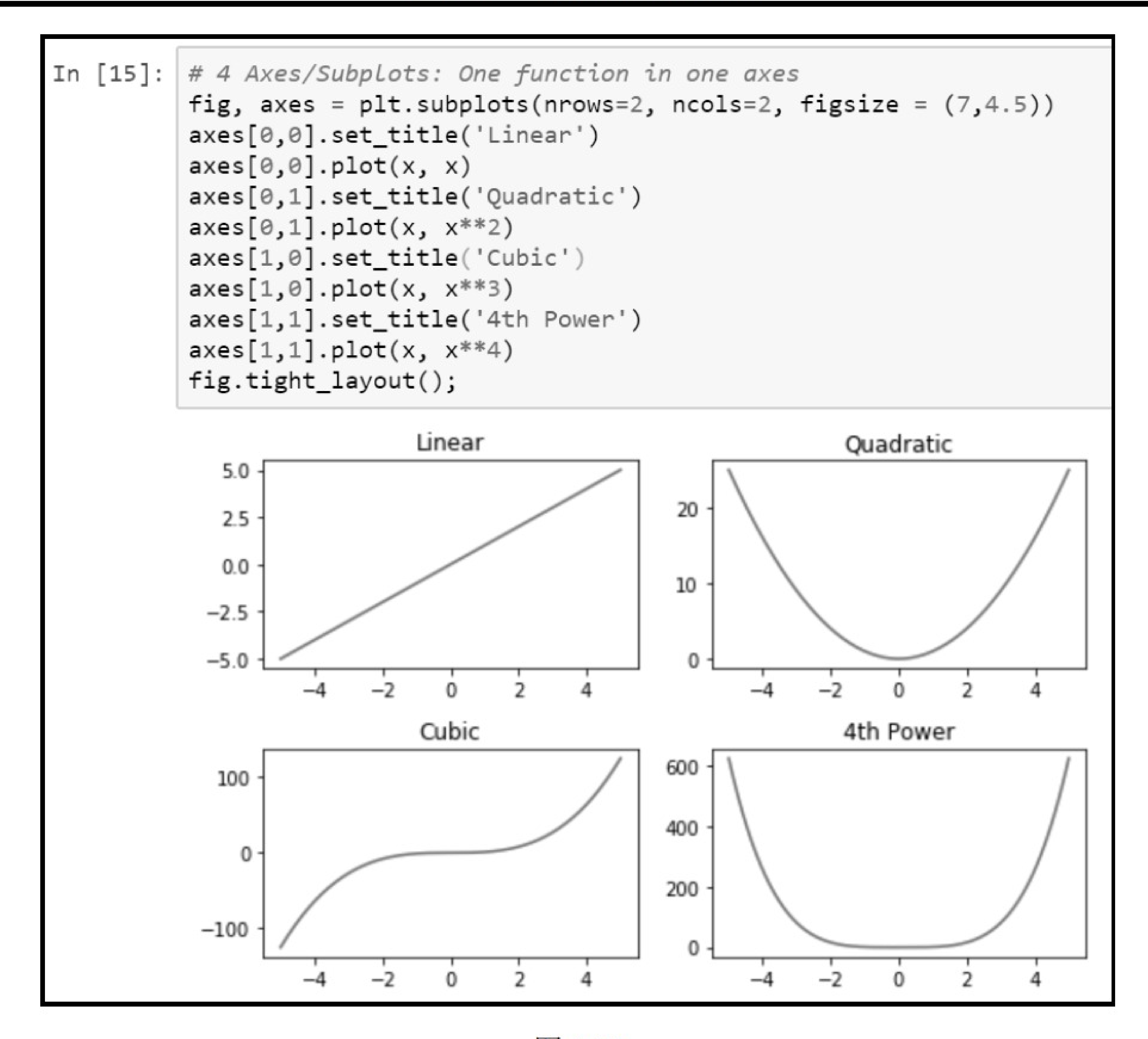

In [15]:
4 Axes/Subplots: One function in one axes
fig, axes = plt.subplots(nrows=2, ncols=2, figsize = (7,4.5))
axes[0,0].set_title('Linear')
axes[0,0].plot(x, x)
axes[0,1].set_title('Quadratic')
axes[0,1].plot(x, x**2)
axes[1,0].set_title('Cubic')
axes[1,0].plot(x, x**3)
axes[1,1].set_title('4th Power')
axes[1,1].plot(x, x**4)
fig.tight_layout();
Linear
Quadratic
Cubic
4th Power

图 4.19

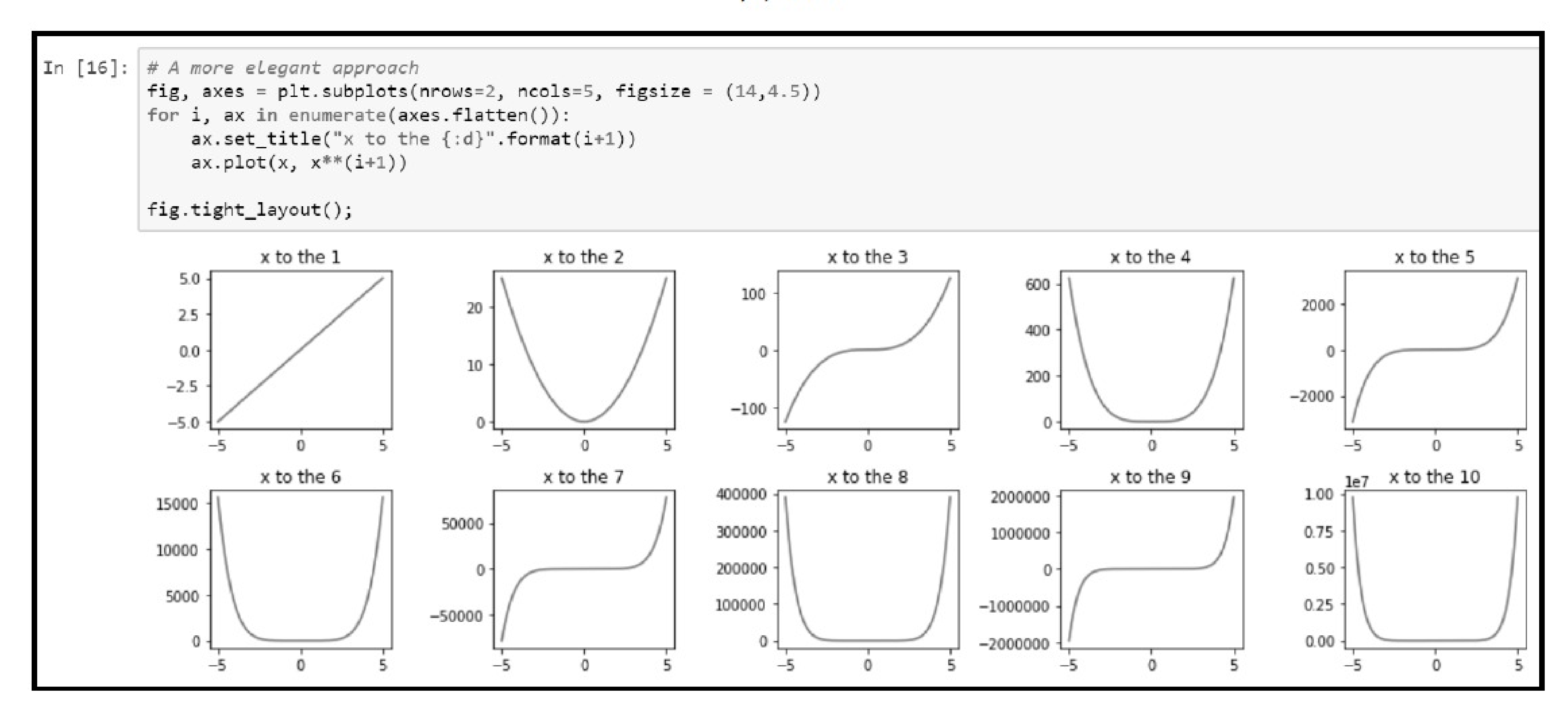

In [16]:
A more elegant approach
fig, axes = plt.subplots(nrows=2, ncols=5, figsize = (14,4.5))
for i, ax in enumerate(axes.flatten()):
    ax.set_title("x to the {:d}".format(i+1))
    ax.plot(x, x**(i+1))
fig.tight_layout();
x to the 1
x to the 2
x to the 3
x to the 4
x to the 5
x to the 6
x to the 7
x to the 8
x to the 9
x to the 10
1e7

图 4.20

4.4　常见的自定义方式

matplotlib 的一个优点是，可对绘图中的单一元素进行调整，当采用 matplotlib 执行数据分析时，常会看到一些较为常用的自定义方式。

下面首先生成一些工作数据，如下所示。

```
# Generating data
x = np.linspace(-np.pi, np.pi, 200)
sine, cosine = np.sin(x), np.cos(x)
```

并在此基础上查看 matplotlib 中的各种自定义特性。

4.4.1　颜色

颜色与绘制窗口中的各种元素均有所关联，matplotlib 对此提供了较好的支持。

matplotlib 中使用颜色最常见的方式是颜色名称或其首字母。因此，对于每种元素，如标题栏、直线或文本的颜色，可传递一个附加的参数，其中包含了对应颜色的字符串。例如，对于蓝色来说，可传递一个字符串 b，或者传递 blue；对于绿色来说同样如此，可传递一个字符串 g，或者传递 green。

下列内容展示了常用的颜色及其首字母列表。

- b：蓝色。
- g：绿色。
- r：红色。
- c：青色。
- m：洋红色。
- y：黄色。
- k：黑色。
- w：白色。

其他允许使用的颜色名则是 HTML/CSS 颜色名，例如 burlywood 和 chartreuse。

注意：

此类颜色名共计 147 个，读者可访问 https://www.w3schools.com/tags/ref_colornames.asp 以了解更多内容。

在 matplotlib 中，另一种颜色的指定方式是使用十六进制数值，这在 HTML 或 CSS

中已有所体现。例如可传递#0000FF 这一类数值，matplotlib 将对此予以支持。

通过传递 0～1.0 的字符串表达方式，还可指定相应的灰度（而非颜色）。其中，0.0 表示为黑色，1.0 表示为白色，其间的每个数值均表示为灰度着色。例如，0.75 表示浅灰色。

除此之外，还可传递 RGB 元组，其中的最后一个数值定义为透明度值，也称作 Alpha。当传递包含 4 个元素的元组时，matplotlib 将前 3 个元素解释为 RGB 颜色，并将最后一个解释为 Alpha 值。因此，假设需要将正弦曲线着色为红色，则可使用 color='red'；而对于余弦曲线，则可使用 color='#165181'，如图 4.21 所示。

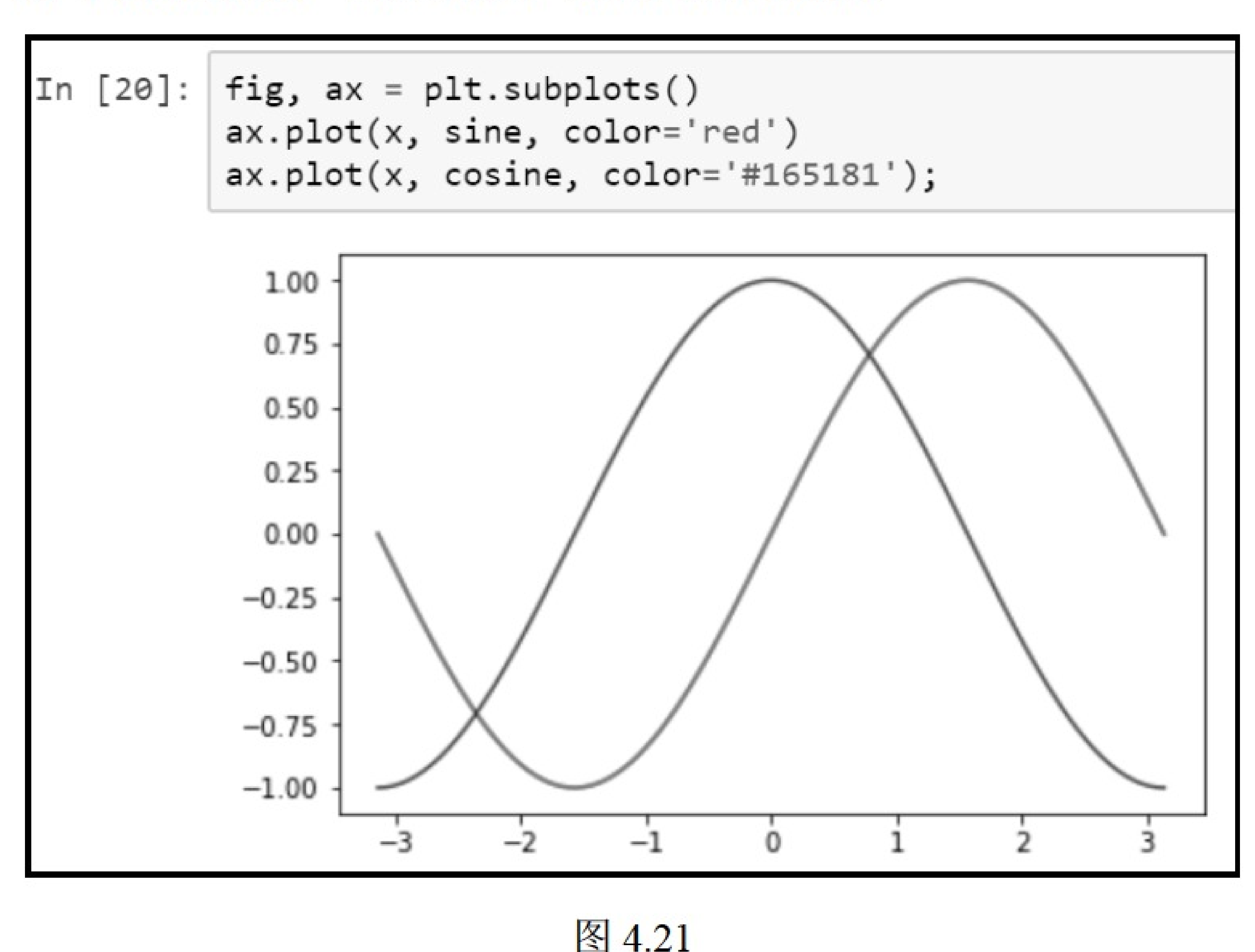


图 4.21

在图 4.21 中可以看到，当运行代码单元时，将分别显示具有不同颜色的曲线。

4.4.2　限定坐标轴

利用 set_xlim()方法，可对 x 轴进行限定；而使用 set_ylim()方法，则可对 y 轴进行限制。此类方法接收两个数值，即最小值和最大值，如图 4.22 所示。

在图 4.22 中可以看到，两个轴向的默认限定条件已发生变化。

4.4.3　设置刻度和刻度标记

matplotlib 可尝试生成相对合理的刻度和刻度标记。这里，刻度是指水平轴向上的标记；而刻度标记或刻度标签则表示与刻度标记对应的数字或数值。另外，根据具体情况，

还可对刻度标记和刻度标签修改，如图 4.23 所示。

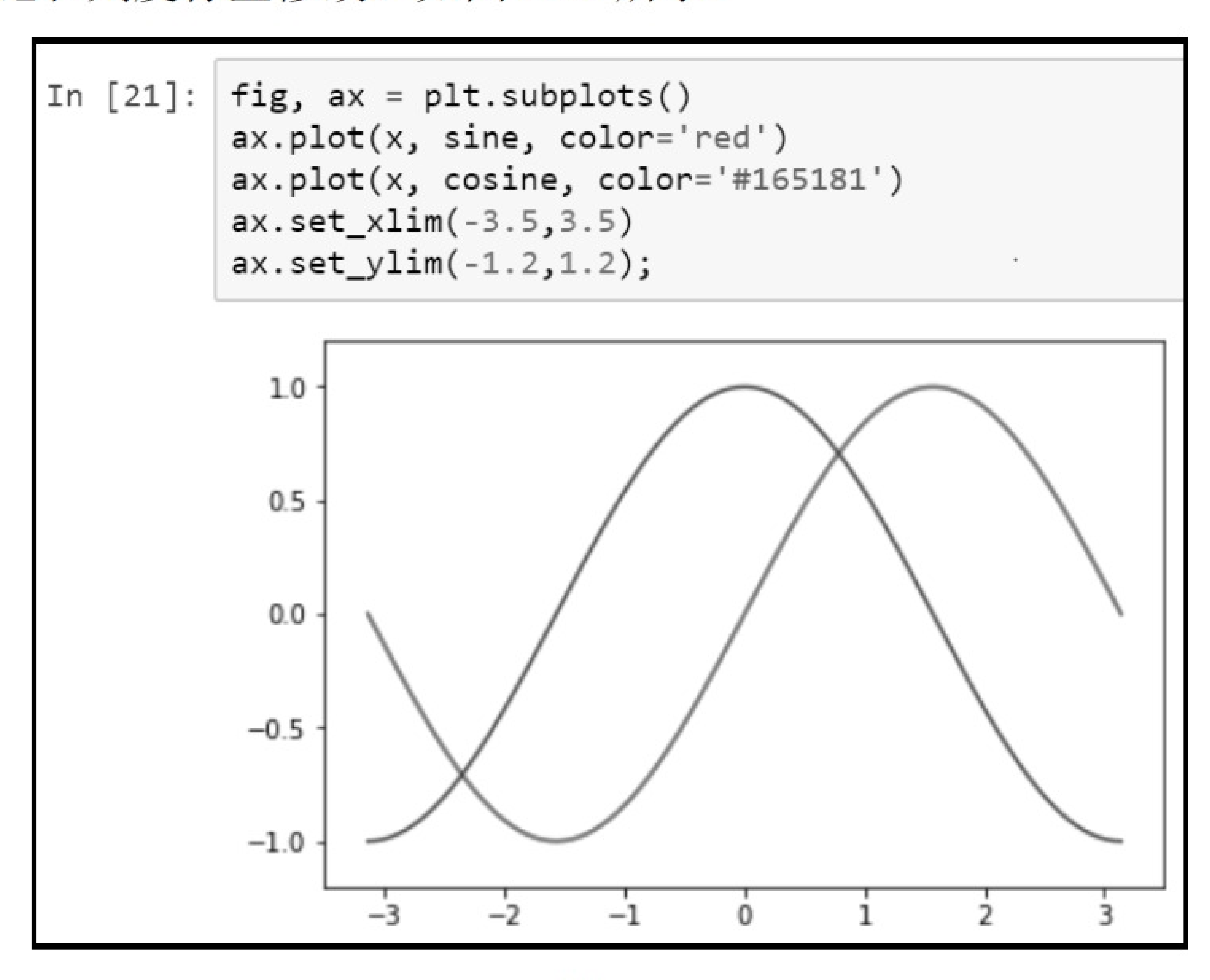


图 4.22

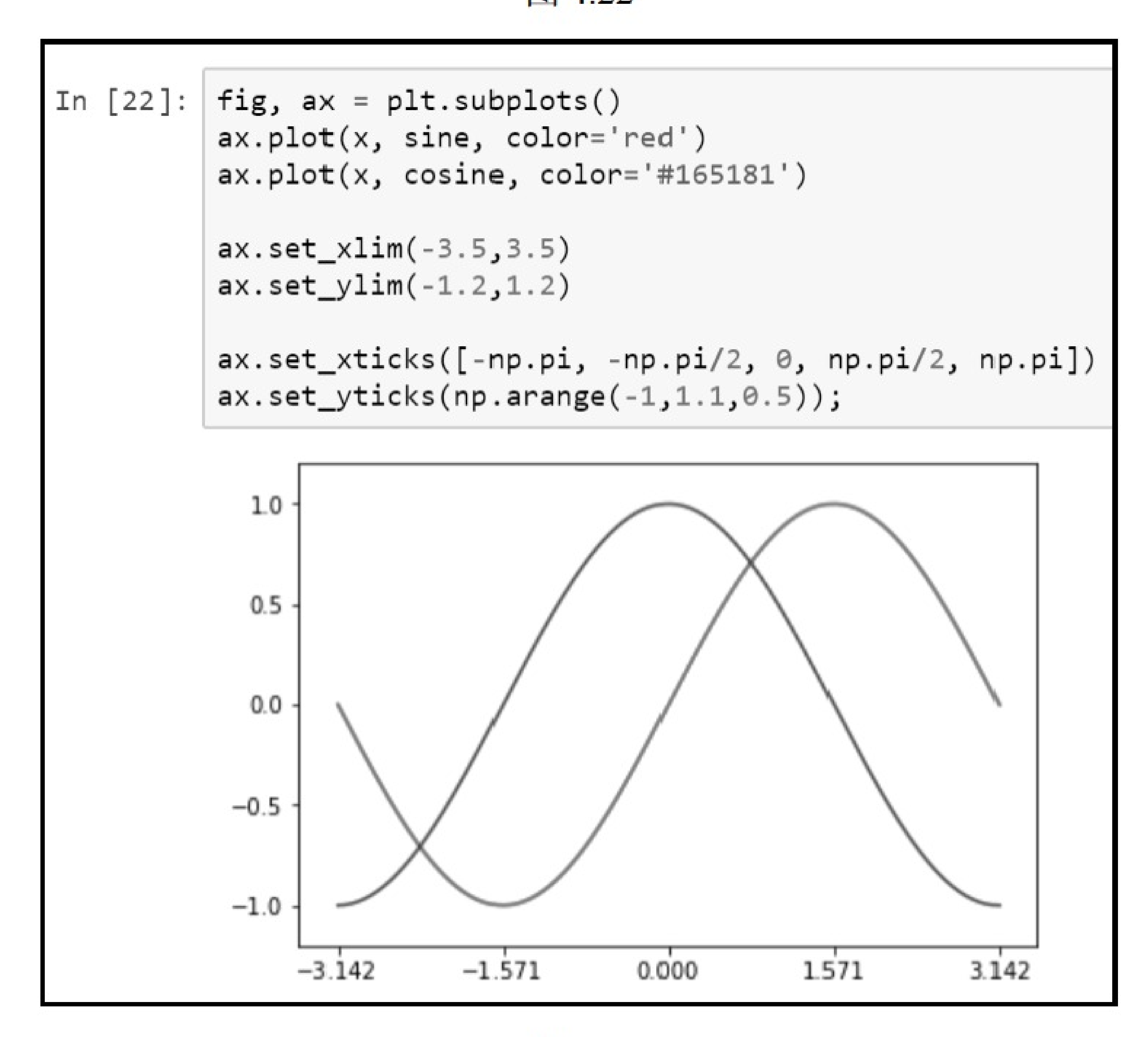


图 4.23

在图 4.23 中，x 轴和 y 轴上包含了一组自定义刻度标记和刻度标签。

类似于修改 xticks 的位置，还可对 xticks 的标签进行调整。假设需要 x 轴上的标记显示为−π 和−π/2 这一类数学符号，则可利用 set_xticks、set_yticks、set_xticklabels 和 set_yticklabels 进行修改并赋予新值，如图 4.24 所示。

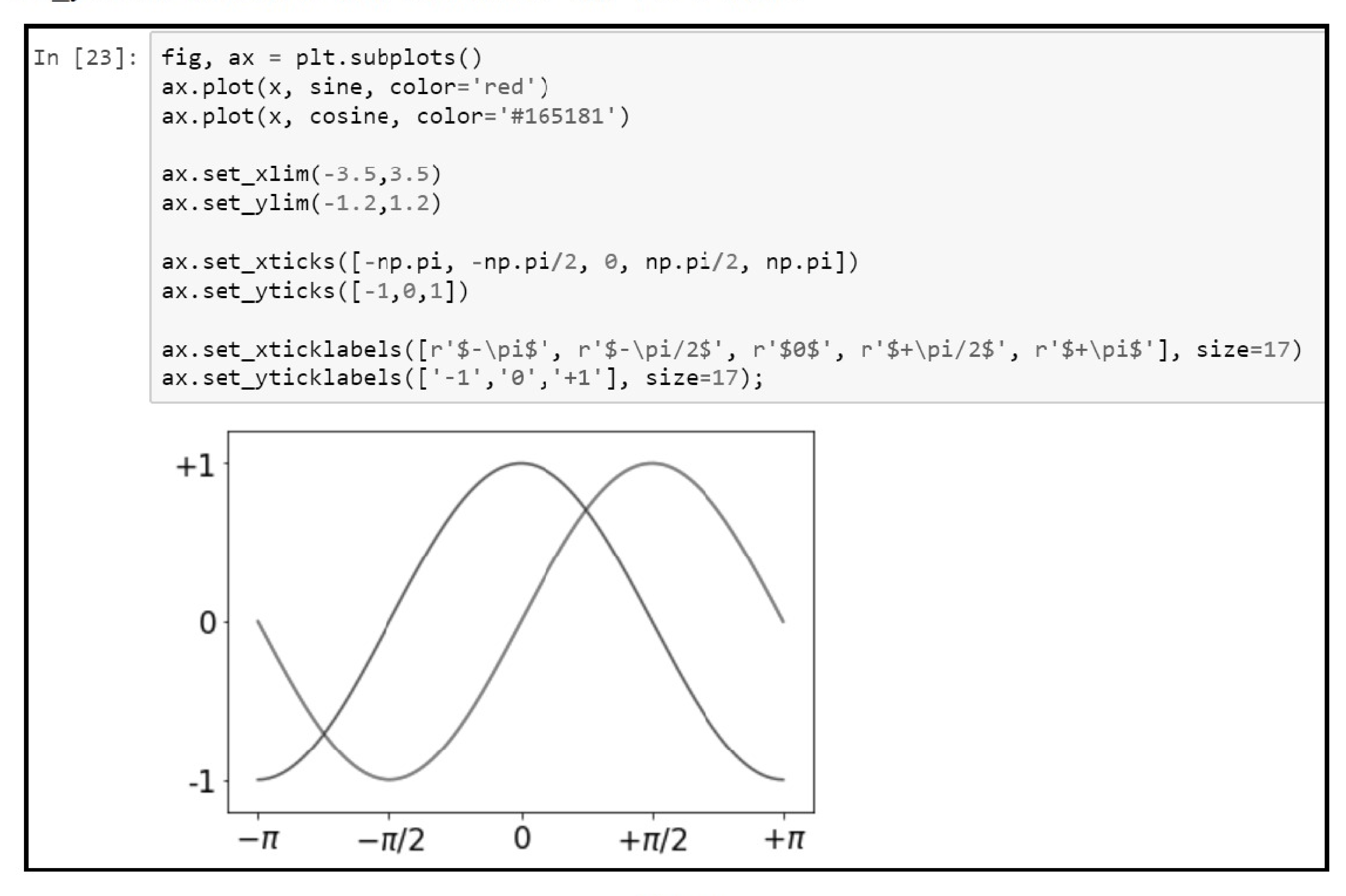


图 4.24

在图 4.24 中，我们修改了刻度标签值，并以数学符号的方式予以呈现。

4.4.4　图例

legend()方法用于生成绘图中的曲线图例，如可辨识正弦曲线和余弦曲线。因此，需要针对所绘制的每条直线指定一个标记。对此，可使用 legend()方法并传递相关参数，进而通知 matplotlib 标记的放置位置，如图 4.25 所示。

在图 4.25 中可以看到，正弦函数和余弦函数图例置于图中的左上方，进而可通过名称方便地识别每条直线。

```
In [24]: fig, ax = plt.subplots()
         ax.plot(x, sine, color='red', label='Sine')
         ax.plot(x, cosine, color='#165181', label='Cosine')

         ax.set_xlim(-3.5,3.5)
         ax.set_ylim(-1.2,1.2)

         ax.set_xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
         ax.set_yticks([-1,0,1])

         ax.set_xticklabels([r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'], size=17)
         ax.set_yticklabels(['-1','0','+1'], size=17)

         ax.legend(loc='upper left');
```


图 4.25

4.4.5　标注

相应地，存在两个主要函数或方法可执行标注工作。具体来说，可通过指定所示文本的 x 坐标和 y 坐标使用 text()方法，该方法的第三个参数为实际绘制的文本内容，如图 4.26 所示。

在图 4.26 中，标注(0,0)通过 ax.text(−0.25,0,'(0,0)')生成；标注(π,0)则通过 ax.text (np.pi−0.25,0, r'$(\pi,0)$', size=15)方法生成。除此之外，annotate 方法则是另一个绘图标注函数。

annotate 方法接收多个参数。针对之前所展示的输出结果，此处可利用 xytext=(1,−0.7)指定文本显示位置；而在 arrowprops=dict(facecolor='blue'))中，我们针对箭头图标传递了一个字典。

```
In [25]: fig, ax = plt.subplots()
         ax.plot(x, sine, color='red', label='Sine')
         ax.plot(x, cosine, color='#165181', label='Cosine')

         ax.set_xlim(-3.5,3.5)
         ax.set_ylim(-1.2,1.2)

         ax.set_xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
         ax.set_yticks([-1,0,1])

         ax.set_xticklabels([r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'], size=17)
         ax.set_yticklabels(['-1','0','+1'], size=17)

         ax.legend(loc='upper left')

         ax.text(-0.25,0,'(0,0)') # x coord, y coord,
         ax.text(np.pi-0.25,0, r'$(\pi,0)$', size=15)

         ax.annotate('Origin',
                     xy=(0, 0), # where the arrow points to
                     xytext=(1, -0.7), # location of text
                     arrowprops=dict(facecolor='blue'));
```


图 4.26

4.4.6　生成网格、水平线和垂直线

当生成网格、水平线和垂直线时，存在多种操作方法可实现这一任务。其中，axhline()方法用于生成水平线；此外，还可进一步针对颜色和透明度指定其他参数，如 Alpha 和颜色值。类似地，axvline()方法则通过相关参数绘制垂直线，对应代码如图 4.27 所示。

grid()方法将生成绘图网格。具体而言，输出结果中的浅灰线使得绘图结果看起来更加舒适。我们几乎可以自定义 matplotlib 中的每个元素，但本章仅涉及较为常用的元素。

```
In [26]: fig, ax = plt.subplots()
         ax.plot(x, sine, color='red', label='Sine')
         ax.plot(x, cosine, color='#165181', label='Cosine')

         ax.set_xlim(-3.5,3.5)
         ax.set_ylim(-1.2,1.2)

         ax.set_xticks([-np.pi, -np.pi/2, 0, np.pi/2, np.pi])
         ax.set_yticks([-1,0,1])

         ax.set_xticklabels([r'$-\pi$', r'$-\pi/2$', r'$0$', r'$+\pi/2$', r'$+\pi$'], size=17)
         ax.set_yticklabels(['-1','0','+1'], size=17)

         ax.legend(loc='upper left')

         ax.text(-0.25,0,'(0,0)') # x coord, y coord,
         ax.text(np.pi-0.2,0.05, r'$(\pi,0)$', size=15)

         ax.annotate('Origin',
                     xy=(0, 0), # where the arrow points to
                     xytext=(1, -0.7), # location of text
                     arrowprops=dict(facecolor='blue'));

         ax.axhline(0, color='black', alpha=0.9) #horizontal line
         ax.axvline(0, color='black', alpha=0.9) #vertical
         ax.grid();
```

图 4.27

4.5　基于 seaborn 和 pandas 的 EDA

探索性数据分析（EDA）是一种数据集分析方案，并对主要特征进行汇总，通常采用可视化方法予以显示。EDA 常用于理解数据、获取与此相关的某些上下文环境、理解变量及其之间的关系，进而形成某种假设条件，以供后续构建预测模式使用。

4.5.1　seaborn 库

seaborn 库可以生成具有吸引力的、信息丰富的图形，其中包括 Python 中的统计信息。

对此，matplotlib 常用于构建 seaborn 库。另外，seaborn 库还可与 Python 数据科学栈集成，同时支持 NumPy、pandas，以及 SciPy 的统计例程和统计模型。

注意：

关于 seaborn 及其特性，读者可访问 https://www.datasciencecentral.com/profiles/blogs/opensource-pythonvisualization-libraries 以了解更多内容。

seaborn 库的导入操作如下。

```
# standard import statement for seaborn
import seaborn as sns
```

4.5.2　执行探索性数据分析

当执行探索性数据分析时，可使用一个数据集，该数据集中包含了美国城市中房屋售价以及户型等信息，我们将把该数据集加载至 pandas DataFrame 中，如图 4.28 所示。

```
In [3]: housing = pd.read_csv('../data/house_train.csv')

In [4]: housing.shape
Out[4]: (1460, 81)

In [5]: housing.info()

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 1460 entries, 0 to 1459
        Data columns (total 81 columns):
        Id                 1460 non-null int64
        MSSubClass         1460 non-null int64
        MSZoning           1460 non-null object
        LotFrontage        1201 non-null float64
        LotArea            1460 non-null int64
        Street             1460 non-null object
        Alley              91 non-null object
        LotShape           1460 non-null object
        LandContour        1460 non-null object
        Utilities          1460 non-null object
        LotConfig          1460 non-null object
        LandSlope          1460 non-null object
        Neighborhood       1460 non-null object
        Condition1         1460 non-null object
        Condition2         1460 non-null object
        BldgType           1460 non-null object
        HouseStyle         1460 non-null object
        OverallQual        1460 non-null int64
```

图 4.28

图 4.28 中的代码也体现了数据集和 pandas DataFrame 间的加载方式，可以看到，DataFrame 中包含了 1460 个观测结果以及 81 列数据。其中，每一列数据体现了 DataFrame 中的一个变量。此处并不打算使用全部变量，相反，我们仅关注执行探索性分析时所涉及的一些变量。

4.5.3　核心目标

所有的数据分析都必须以一些关键问题或目标为指导，在开始执行数据分析任务之前，我们的头脑中须建立起一个清晰的目标。作为示例，以下内容显示了数据集探索性数据分析过程中的主要目标。

- 理解数据集中的单一变量。
- 理解数据集中变量与房屋售价之间的关系。

只有在深入理解了数据和当前问题后，方可得到有意义的分析结果。

4.5.4　变量类型

整体上讲，变量存在以下两种可能的数据类型。

- 数值变量。
- 类别变量。

数值变量表示相关变量值为数字；而类别变量则意味着对应值为相关分类，如图 4.29 所示。

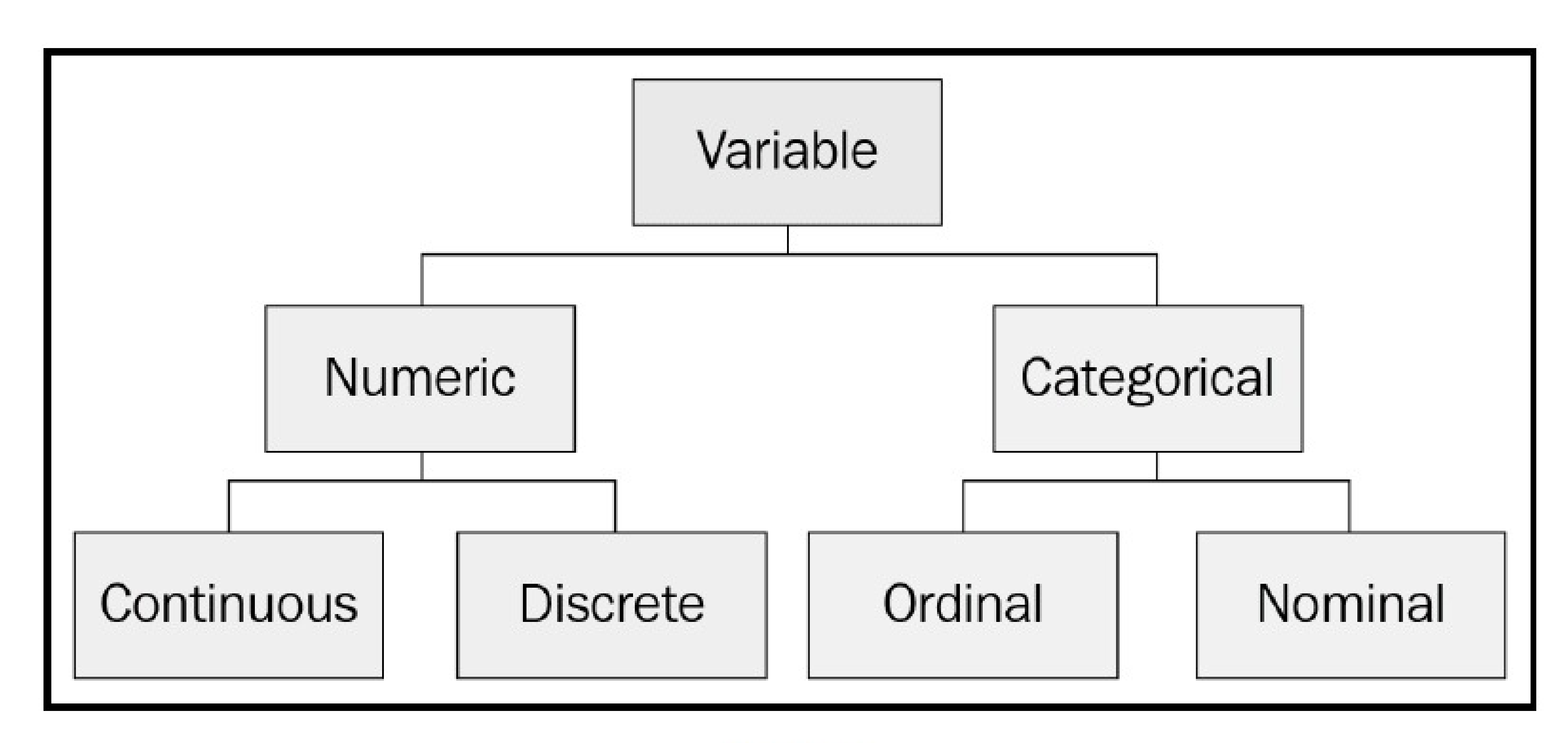


图 4.29

上述两个较大的分类中各包含了两个子分类。对于数值变量，一方面，定义了连续变量，连续变量理论上可以取区间内的任何值。另一方面，离散数字变量可表示为区间

内的某些特定值；对于类别变量，其中涵盖了两种类别变量类型。第一种变量定义为有序变量，且包含了对应类别的自然顺序。例如，如果定义了一个 quality 变量，该变量的分类对应于低质、中质、优质，鉴于该分类中包含了自然顺序，因而可得到一个有序变量。在某种意义上讲，中质优于低质；而优质则胜于中质和低质。第二种变量为名义（Nominal）类别变量，是指类别变量不包含任何顺序。

下面考查数据集中的数值变量示例。

- SalesPrice：该变量表示房屋的售价。
- LotArea：该变量表示房屋面积。
- OverallQual：该变量表示整体材料的利用率。
- OverallCond：该变量代表房屋的整体状况。
- 1stFirSF：该变量表示一楼的面积。
- 2ndFirSF：该变量表示二楼的面积。
- BedroomAbvGr：该变量表示楼上的卧室（不包括地下室的卧室）。
- YearBuilt：变量表示最初的构建日期（从技术上讲，这不是一个数值变量，但是我们将使用它来生成另一个名为 Age 的变量）。

下面考查数据集中与类别变量相关的一些示例。

- MSZoning：该变量用于标识销售的一般分区分类。
- LotShape：该变量表示房屋的户型。
- Neighborhood：该变量表示 Ames 城市范围内的物理位置。
- CentralAir：该变量表示中央空调系统。
- SaleCondition：该变量表示出售条件。
- MoSold：该变量表示出售月份（MM）。
- YrSold：该变量表示出售年份（YYYY）。

后面在实际分析数据时，将会看到为什么这些变量被称作数值变量和类别变量。

4.6　单独分析变量

首先需要定义分析过程中所用的变量名，当前包含了一个数值变量列表和类别变量列表。随后，将重新定义房屋的 DataFrame，其中仅包含了刚刚定义的变量。接下来，可使用 shape 属性查看新 DataFrame 的大小，如图 4.30 所示。

在图 4.30 中，可以看到 DataFrame 的尺寸发生了变化，且仅包含 15 列。

```
In [7]: numerical_vars = ['SalePrice','LotArea', 'OverallQual', 'OverallCond',
                          'YearBuilt', '1stFlrSF', '2ndFlrSF', 'BedroomAbvGr']
        categorical_vars = ['MSZoning', 'LotShape', 'Neighborhood', 'CentralAir', 'SaleCondition', 'MoSold', 'YrSold']

In [8]: housing = housing[numerical_vars+categorical_vars]

In [9]: housing.shape
Out[9]: (1460, 15)
```

图 4.30

4.6.1　理解主变量

下面介绍一下主变量，即房屋的 SalePrice。对于一个类别变量，我们要做的第一件事就是了解其描述性统计数据，如图 4.31 所示。

```
In [10]: #descriptive statistics summary
         housing['SalePrice'].describe()

Out[10]: count       1460.000000
         mean      180921.195890
         std        79442.502883
         min        34900.000000
         25%       129975.000000
         50%       163000.000000
         75%       214000.000000
         max       755000.000000
         Name: SalePrice, dtype: float64
```

图 4.31

至此，我们已了解到主变量的取值范围。在图 4.31 中，可以看到数据集中的平均房价为 180000 美元。另外，标准偏差约为 80000 美元。与数据集中最便宜的房子对应的最小值约为 35000 美元；与数据集中最昂贵的房子对应的最大值为 755000 美元。

一般来讲，获取计算数值变量的描述性统计是一种较好的习惯，进而了解该变量可取的数值，以及该变量的分布和离散状态。除此之外，还需要通过变量的图形表达进一步完善从数值汇总中获取的信息。对于数值变量，一种典型的变量可视化理解方式是柱状图。当显示 pandas Series 的柱状图时，可使用 hist 方法，如图 4.32 所示。

从图 4.32 中可以看出，房屋的价格很少低于 100000 美元，且大多数房屋的价格集中在 100000 美元～200000 美元。此外，图中仅包含了少量的、价格过高的观测结果。也就是说，很少有超过 400000 美元的房屋，且图中的分布结果呈现为长尾状态，这一点可通

过 Skewness 和 Kurtosis 数值统计得到确认。其中，偏度为 0 表示正态分布；对于正值，将得到一个与当前显示类似的正态尾部状态。相应地，峰度则表示分布的厚度。在图 4.32 中，较高的数值主要集中在 100000 美元～200000 美元——据此，当前峰度为 6.5。接近正态分布的变量其值通常在 3 左右。

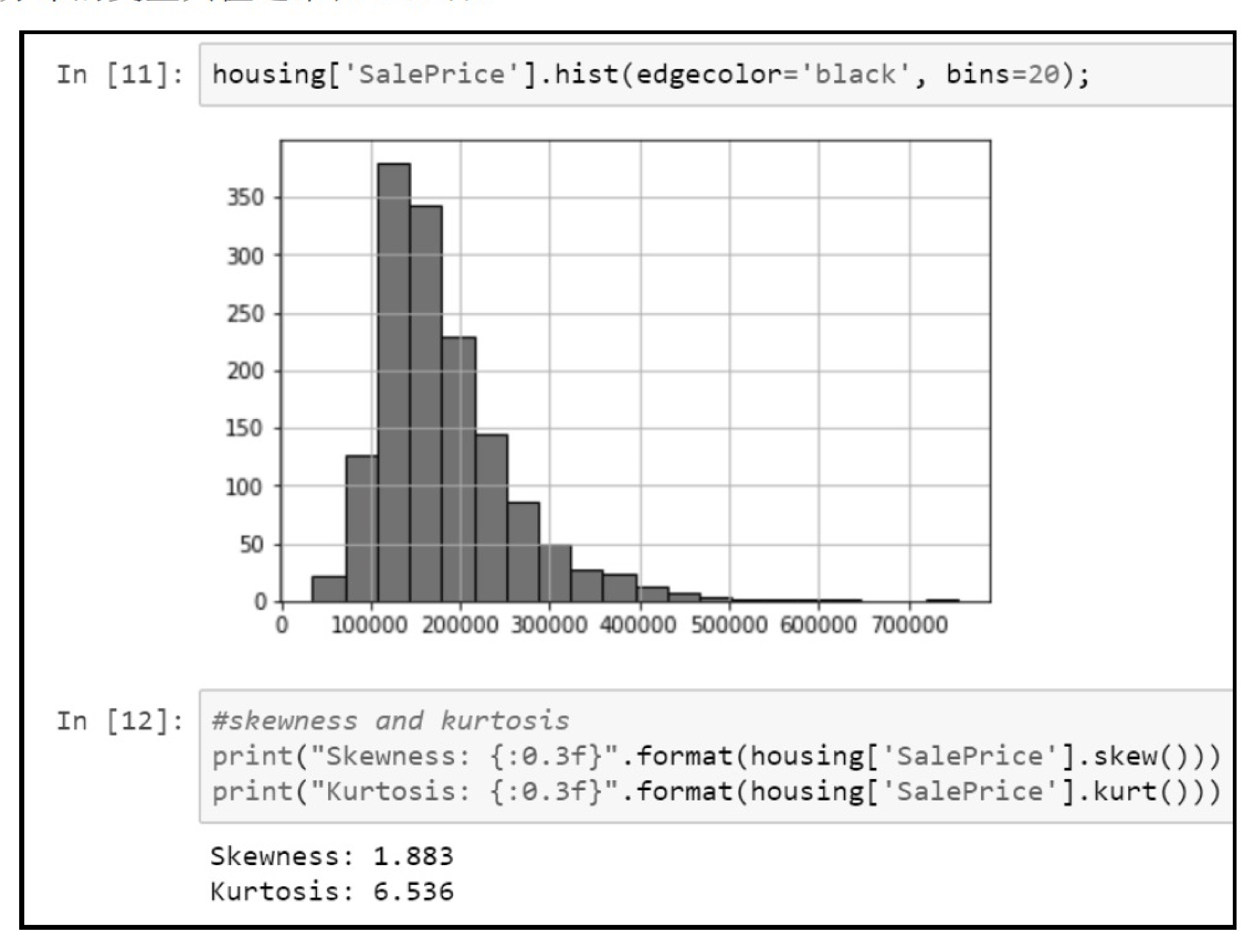
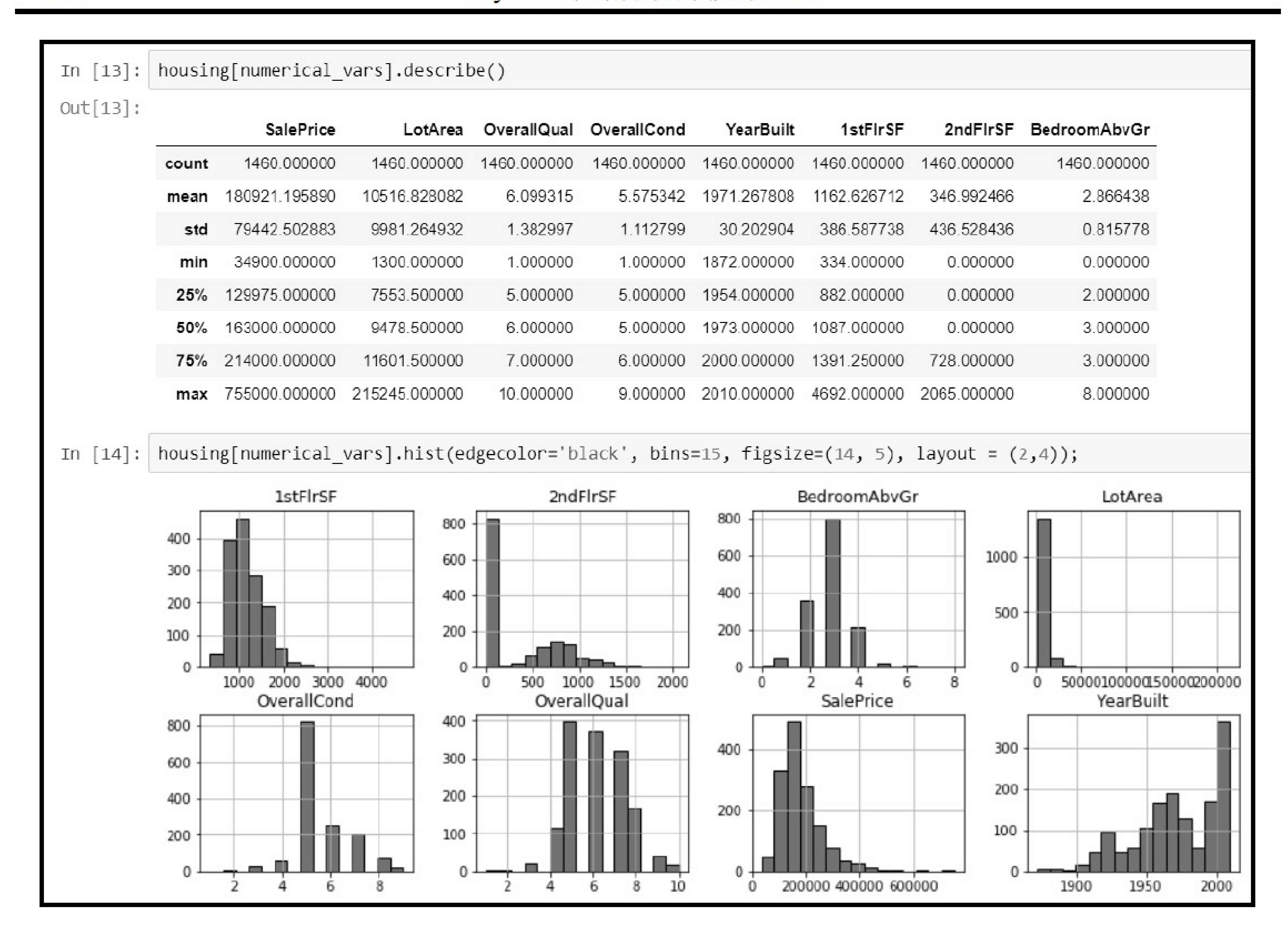


图 4.32

4.6.2　数值变量

如果希望检测数据集中的全部变量，可以非常方便地使用针对 pandas DataFrame 对象获取的 hist()方法来实现。对此，可利用现有的数值变量列表构建房屋 DataFrame 的子集，同时，正如我们对 pandas Series 所做的那样，也可以对 pandas DataFrame 执行同样的操作。因此，可使用 hist()方法并设置以下参数：edgecolor 赋予为黑色，以使每个柱状图案之间的直线显示为黑色；柱状图中的 bins 数量则设置为 15；figsize 设置为(14,5)；layout 设置为 2 行、4 列。图 4.33 共计显示了 8 个柱状图。

pandas 针对数据集中 8 个数值变量生成了较好的可视化效果，据此，我们可以达到大量的有用信息，具体如下。



In [13]:
```
housing[numerical_vars].describe()
```

Out[13]:

	SalePrice	LotArea	OverallQual	OverallCond	YearBuilt	1stFlrSF	2ndFlrSF	BedroomAbvGr
count	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
mean	180921.195890	10516.828082	6.099315	5.575342	1971.267808	1162.626712	346.992466	2.866438
std	79442.502883	9981.264932	1.382997	1.112799	30.202904	386.587738	436.528436	0.815778
min	34900.000000	1300.000000	1.000000	1.000000	1872.000000	334.000000	0.000000	0.000000
25%	129975.000000	7553.500000	5.000000	5.000000	1954.000000	882.000000	0.000000	2.000000
50%	163000.000000	9478.500000	6.000000	5.000000	1973.000000	1087.000000	0.000000	3.000000
75%	214000.000000	11601.500000	7.000000	6.000000	2000.000000	1391.250000	728.000000	3.000000
max	755000.000000	215245.000000	10.000000	9.000000	2010.000000	4692.000000	2065.000000	8.000000

In [14]:
```
housing[numerical_vars].hist(edgecolor='black', bins=15, figsize=(14, 5), layout = (2,4));
```


图 4.33

- ❑ 从 1stFlrSF 中可以看到，一层的面积分布状态向右倾斜。也就是说，大多数房屋的面积基本在 1000 平方英尺[①]或 1200 平方英尺左右。
- ❑ 在 2ndFlrSF 变量中，最大的柱状图大约为 0，表明大多数房屋均不包含二层。
- ❑ 大多数房屋均配置了 3 间卧室。
- ❑ 在 LotArea 中，柱状图高度倾斜，也就是说，面积大的户型较少。
- ❑ 条件和质量评级结果大约为 5，这说明大部分房屋的评级适中（较高和较低的评级较少）。
- ❑ 当前，YearBuilt 变量实际上并无太多用处。但是，我们可以以此构建一个有意义的变量（出售时房屋的使用时间）。

因此，需要定义新的变量 Age，即服务出售年份减去其建筑年份。然后，可从数值变量中移除 YearBuilt 变量，并利用数值变量列表中的 Age 变量予以替换。当再次进行绘制时，Age 的分布状态如图 4.34 所示。

[①] 图中用的单位是平方英尺，书中单位与图中单位保持一致。

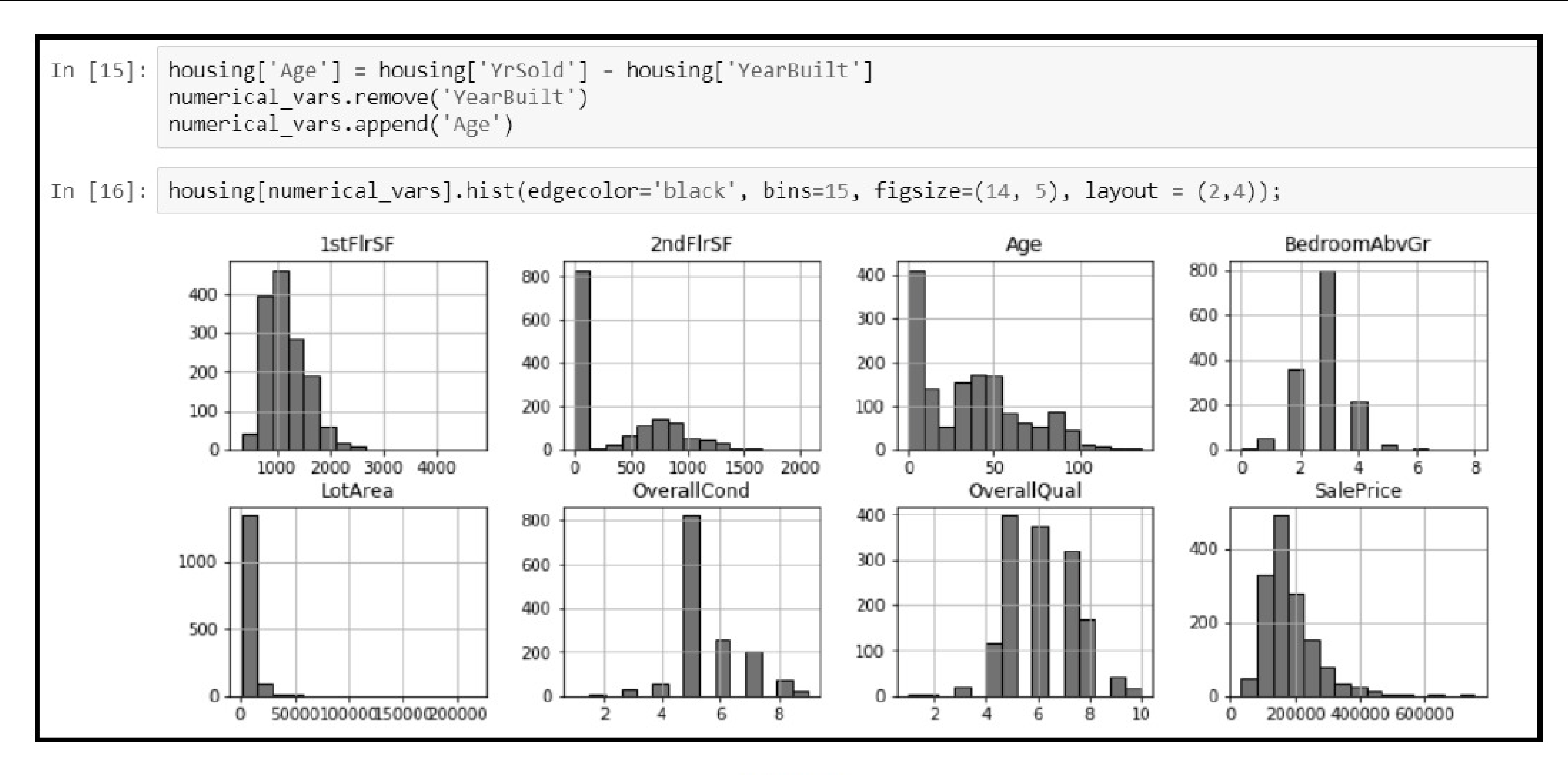


图 4.34

从图 4.34 中可以看到，包含较大柱状图的 Age 变量位于 0 处，这也表明，大量的房屋已被售出，且大约 400 套新房子已被售出。

4.6.3 类别变量

柱状图是类别变量的推荐绘图类型。当在 pandas 中针对类别变量绘制柱状图时，可使用 pandas Series 对象 SaleCondition，计算 value_counts 并于随后使用 plot()方法，如图 4.35 所示。

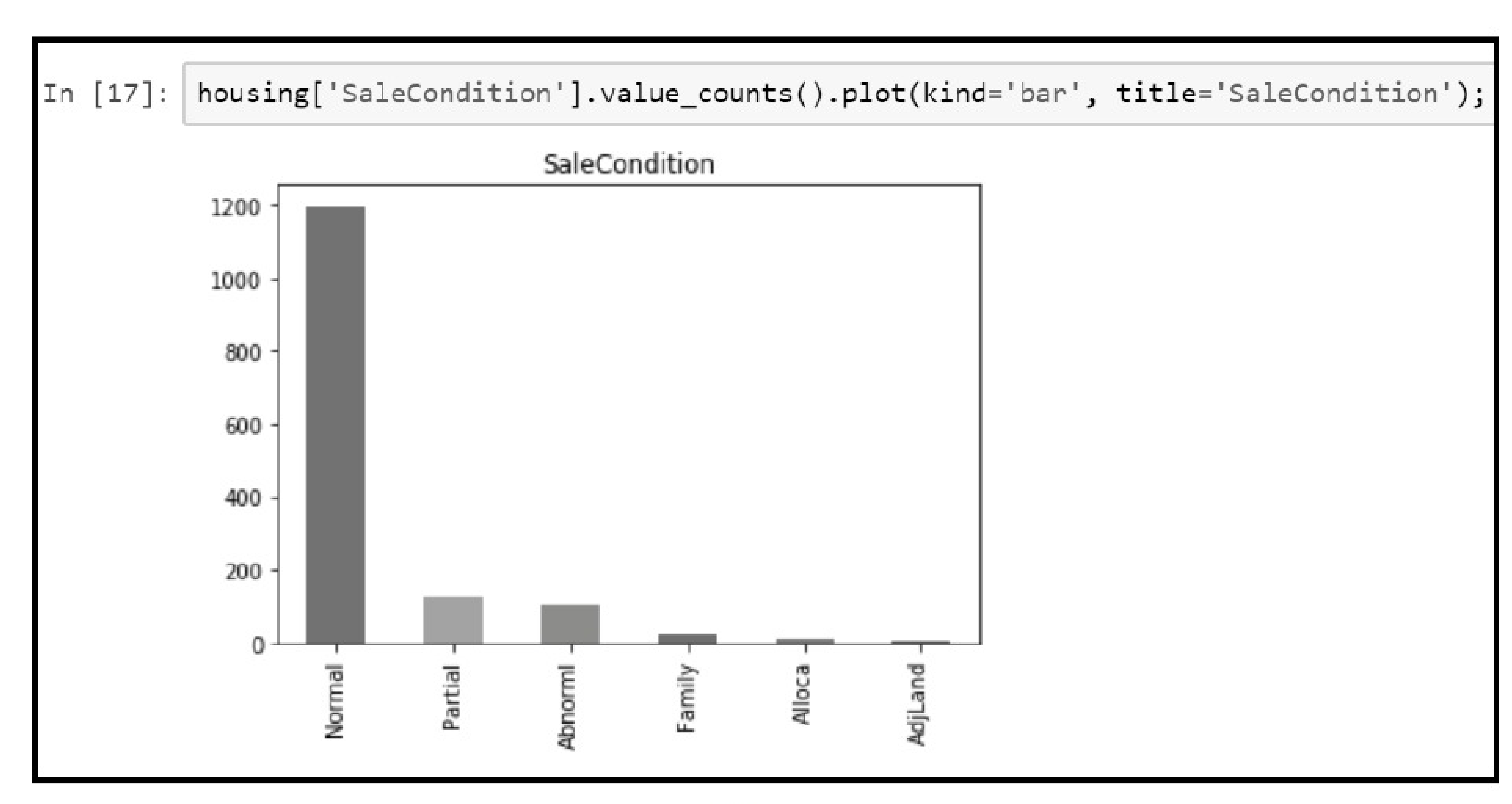


图 4.35

当执行图 4.35 中的代码行时，将得到 SaleCondition 变量中不同类别的计数结果。因此，大多数房屋在 Normal 条件下即被出售，其中数量约为 1200 套；少量房屋则在其他条件下被出售。

当对数据集中的全部类别变量执行可视化操作时，类似于数值变量，并不存在一种方法可直接利用 pandas 对其进行操作。对此，可通过之前所学知识实现这一任务（参见 4.1 节）。

因此，可利用 plt.subplot 函数生成一个绘制窗口和一个轴对象，即 2 行 4 列的网格，共计 8 个子图。随后，可将 figsize 设置为(14,6)。接下来，可编写一个循环语句，用于遍历 ax.flatten()对象中每个子图的各个类别变量。

对于每个变量，可采用 pandas Series 对象计算数值计数结果。随后可使用 plot()方法并将 kind 设置为 bar。这里，唯一的调整是绘制 subplot 对象和 loop 变量中的柱状图。最后，可针对子图调用 tight_layout，以实现较好的输出效果，如图 4.36 所示。

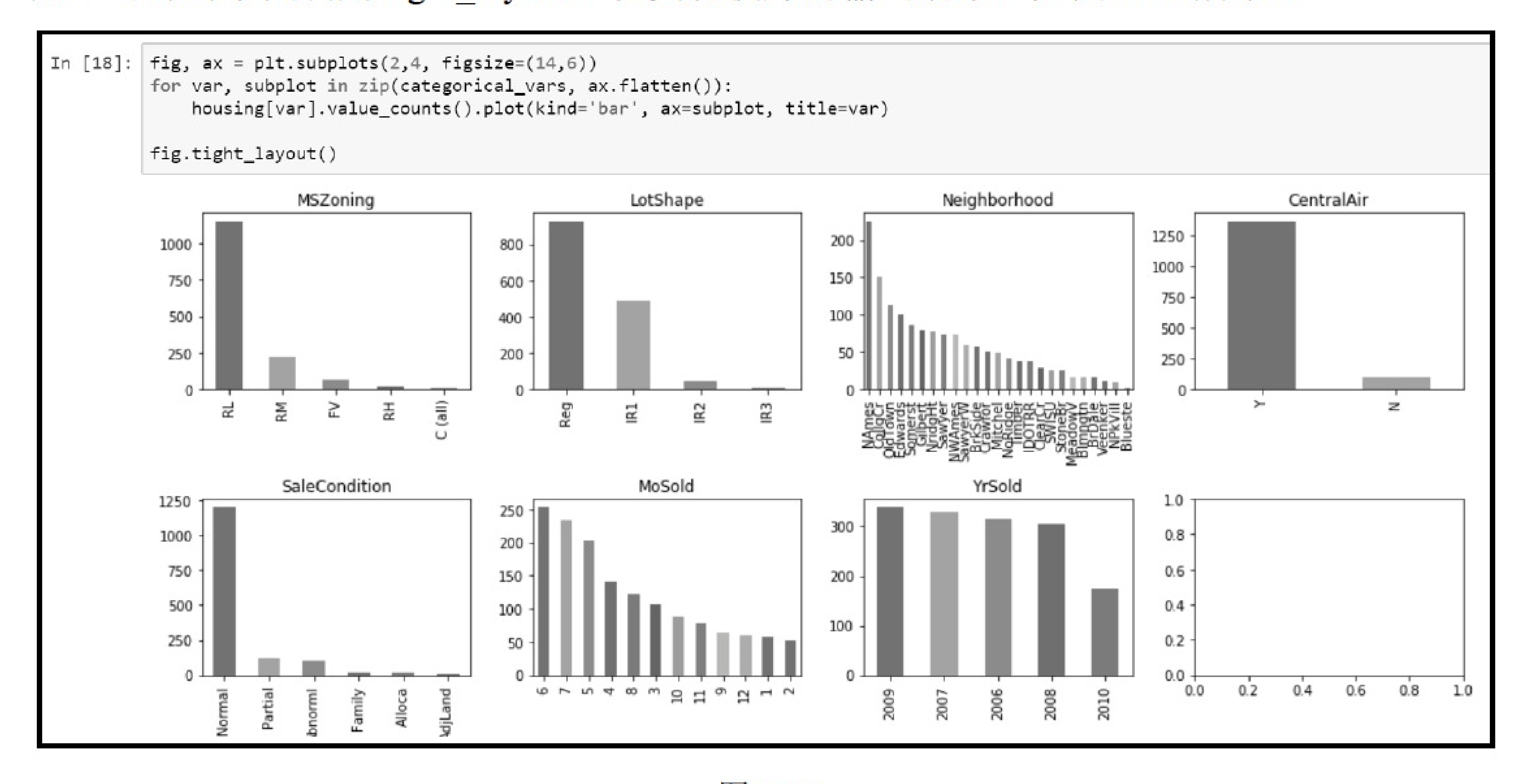


图 4.36

可以看到，通过图 4.36 中的 4 行代码，即可生成相对复杂且内容丰富的可视化效果。

为了更好地理解变量和 SalePrice 之间的关系，可编写一个小型函数，以识别小于 30 个数值的分类级别。随后，可通过 apply()方法将该函数应用于 pandas DataFrame 中的各行。接下来，通过一条 for 循环语句检测并保存大于 30 个观测结果的分类级别，如图 4.37 所示。

```
In [19]: def identify_cat_above30(series):
             counts = series.value_counts()
             return list(counts[counts>=30].index)

In [20]: levels_to_keep = housing[categorical_vars].apply(identify_cat_above30, axis=0)
         levels_to_keep

Out[20]: MSZoning                                                [RL, RM, FV]
         LotShape                                              [Reg, IR1, IR2]
         Neighborhood      [NAmes, CollgCr, OldTown, Edwards, Somerst, Gi...
         CentralAir                                                    [Y, N]
         SaleCondition                             [Normal, Partial, Abnorml]
         MoSold                         [6, 7, 5, 4, 8, 3, 10, 11, 9, 12, 1, 2]
         YrSold                                 [2009, 2007, 2006, 2008, 2010]
         dtype: object

In [21]: for var in categorical_vars:
             housing = housing.loc[housing[var].isin(levels_to_keep[var])]

In [22]: housing.shape

Out[22]: (1246, 16)
```

图 4.37

当执行上述代码时，将会得到少量的观测结果，即 16 个变量的 1246 个观测结果。下面再次查看之前显示的可视化结果，如图 4.38 所示。

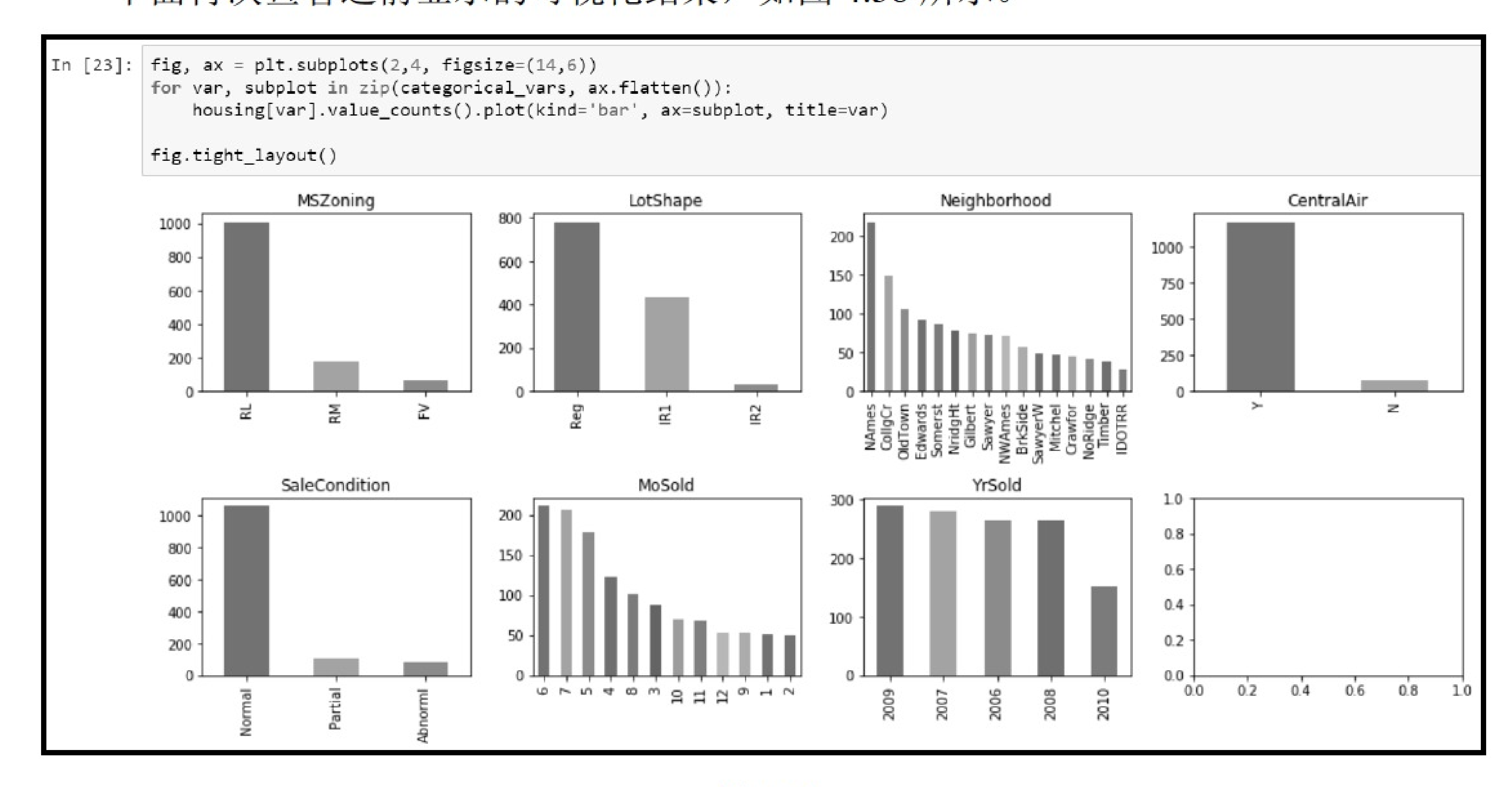


图 4.38

可以看出，图 4.38 中并未包含小于 30 个观测结果的分类级别。当前，MSZoning 变量、LotShape 变量、SaleCondition 变量均包含了 3 个分类级别。

4.7　变量间的关系

不同变量间的关系可通过 matplotlib 的命名空间图实现可视化效果。其中，散点图常用于可视化两个数值变量间的关系；箱形图用于数值变量和类别变量间的可视化关系；复杂条件图则用于多个变量的可视化效果。

4.7.1　散点图

当利用 pandas 生成散点图时，全部所需工作是使用图命名空间。在图命名空间内，可采用 scatter()方法并传递 x 和 y 值，如图 4.39 所示。

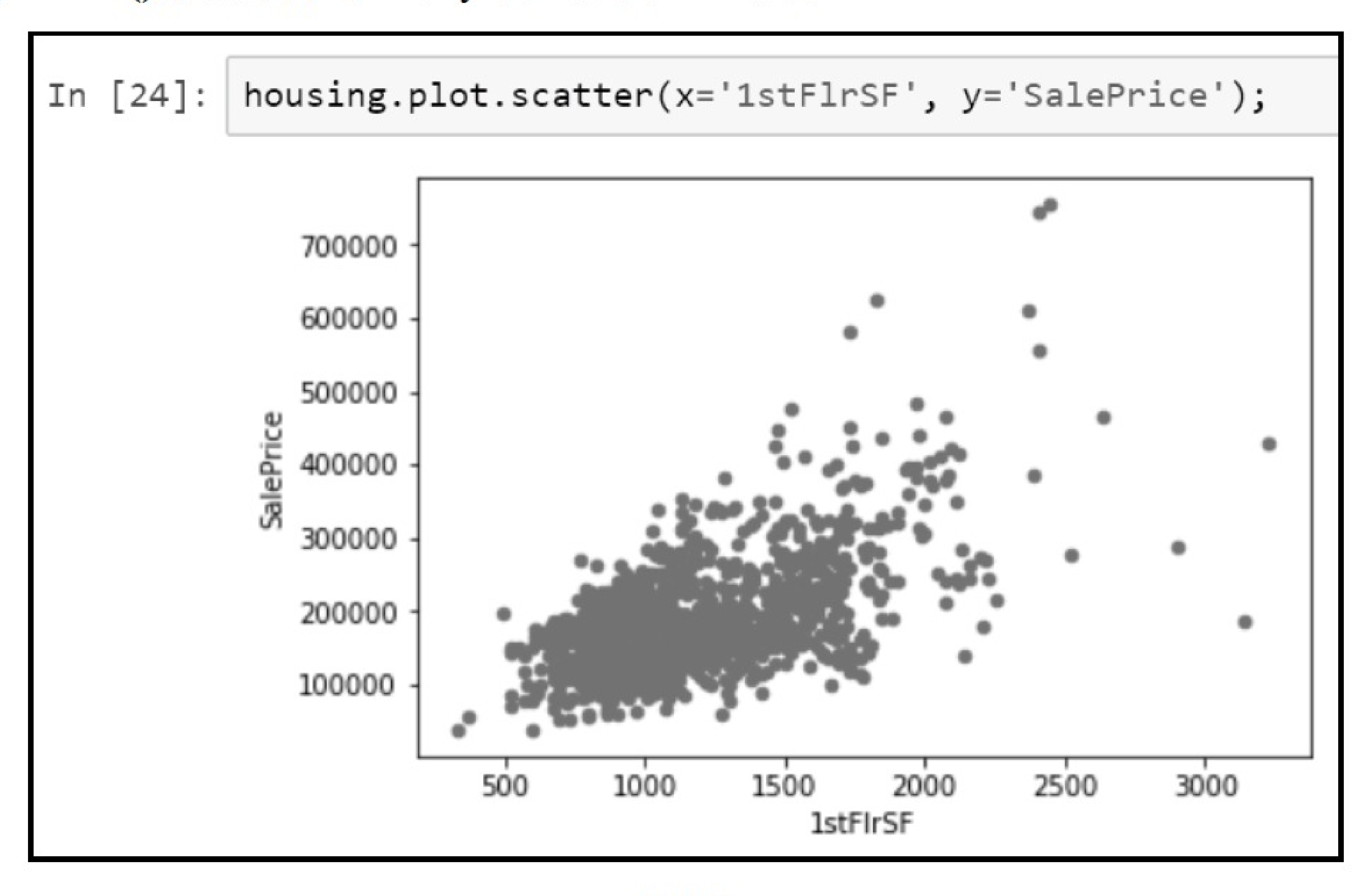


图 4.39

从图 4.39 中可以看出，1stFlrSF 和 SalePrice 之间具有正关系。因此，变量 1stFlrSF 绘制于 x 轴上，而变量 SalePrice 绘制于 y 轴上。图中可清晰地看到，两个变量间存在正关系——变量 1stFlrSF 越大，则房屋的销售价格就越高。

seaborn 提供了一个名为 jointplot 的函数，除了能够生产散点图之外，该函数还可生成边际图，如图 4.40 所示。

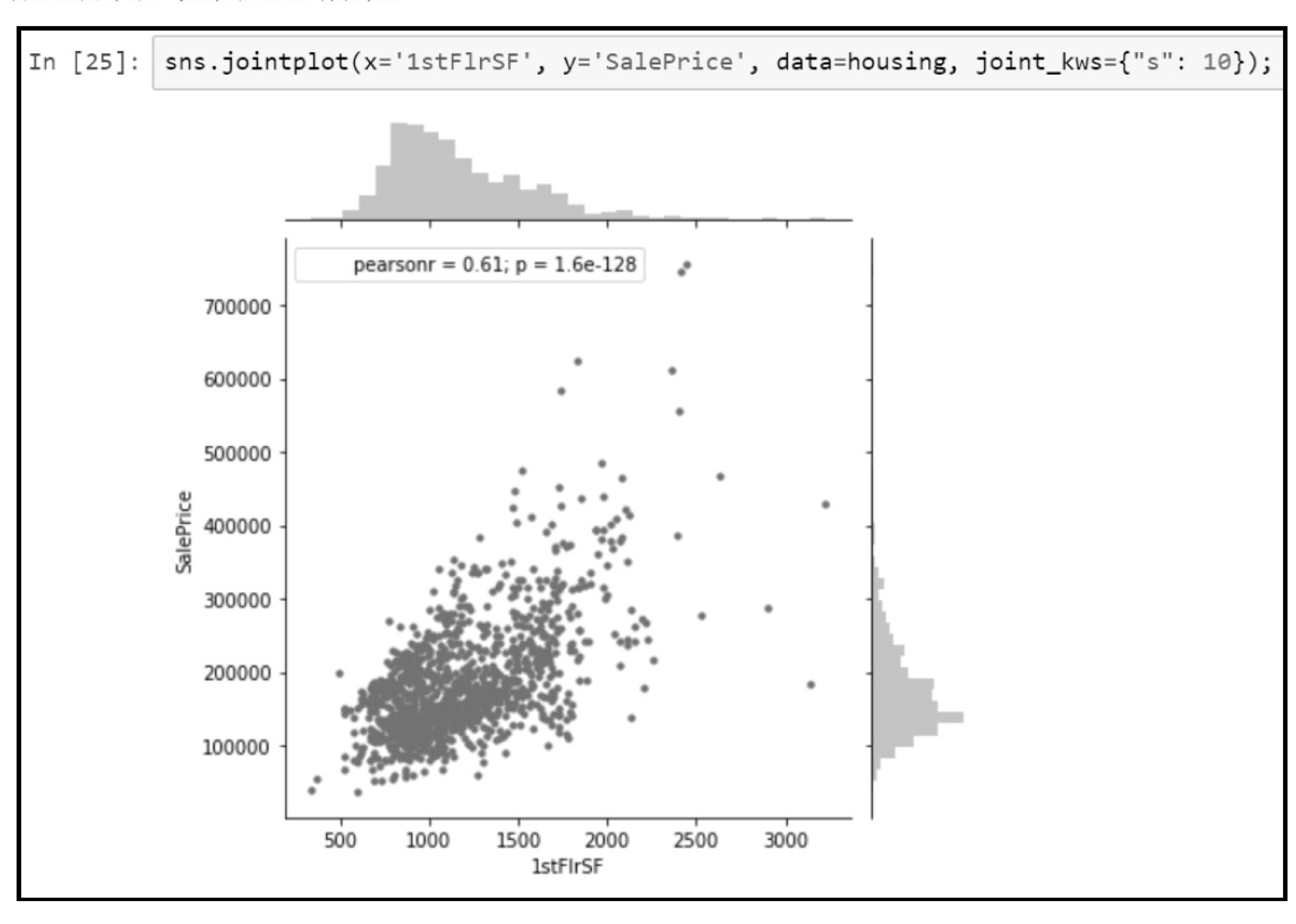


图 4.40

在图 4.40 所示的边际图中，可以看到变量在 x 轴和 y 轴上的分布状态。据此，不仅可以查看两个变量间的关系，还可进一步观察到变量各自的分布方式。

如果打算同时显示多幅散点图，可创建一个散点图矩阵，利用 pairplot 函数，seaborn 可方便地实现这一功能。对此，可传递一个包含数值变量的 DataFrame，对应结果即为散点图矩阵，如图 4.41 所示。

其中，每幅散点图均体现了 DataFrame 中变量间的相互关系。建议同时使用不超过 4 个或 5 个变量，否则，可视化效果将难以得到完美地展示。

此处采用了 4 个变量，其中之一是房屋的 SalePrice。不难发现，SalePrice 和另一个变量间存在正关系。随后，LotArea 显示了 OverallQual 和 SalePrice 间的关系。这一关系仍呈现为正关系，但不如 OverallCond 那样明显。

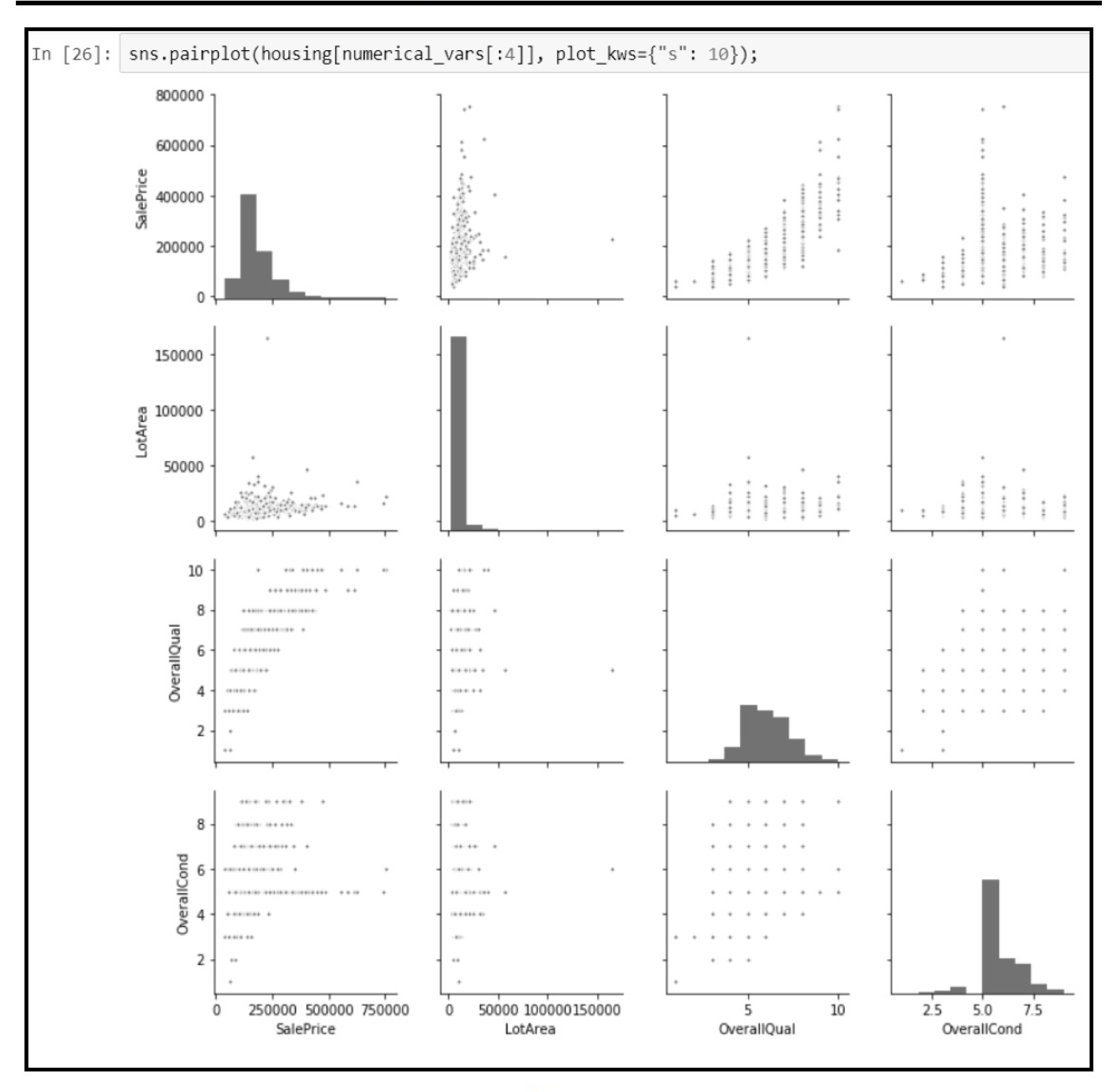


图 4.41

图 4.42 显示了多个数值变量与 SalePrice 之间的关系。

通过观察可知，SalePrice 与 1stFlrSF 和 2ndFlrSF 之间具有较为明显的正关系。此外，Age 和 SalePrice 变量间具有非线性的负关系。具体来说，随着 Age 增加，SalePrice 呈现下降趋势，因而这一关系表示为曲线而非直线。

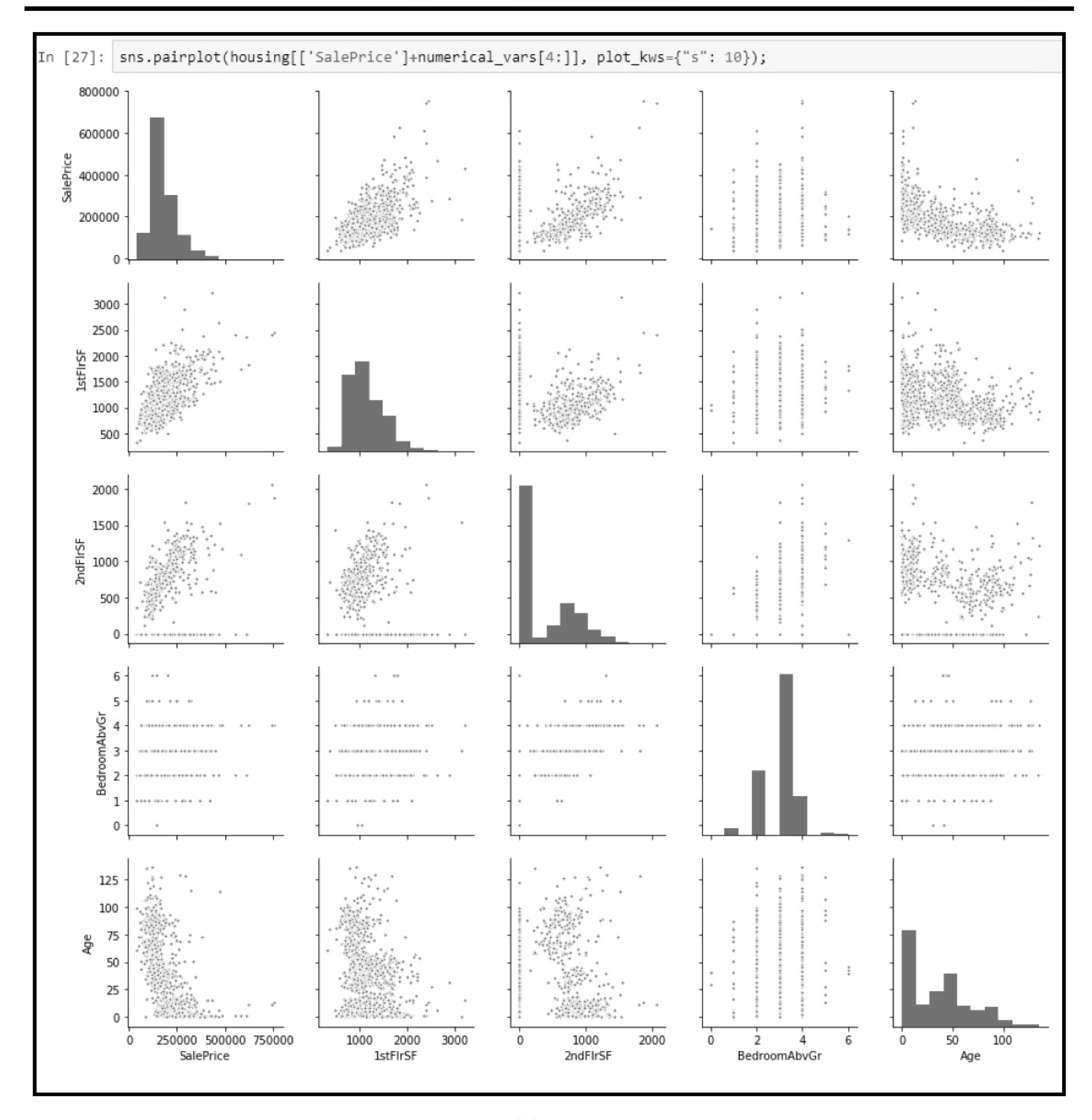


图 4.42

4.7.2　箱形图

这里将考查数据集中的类别变量和房屋 SalePrice 之间的关系。对此，箱形图（也称箱须图）则是查看数值和类别变量关系的一种标准可视化方式。箱形图通过其四分位数

表述一组数字。另外，箱形图也包含垂直方向上扩展的直线，表示上、下四分位数之外的变化。其中，某些异常值可以绘制为单独的点。箱形图是非参数化的，并显示了统计总体样本的变化，而没有对潜在的统计分布做出任何假设。

下面通过箱形图考查 alePrice 与其他类别变量之间的关系，如图 4.43 所示。

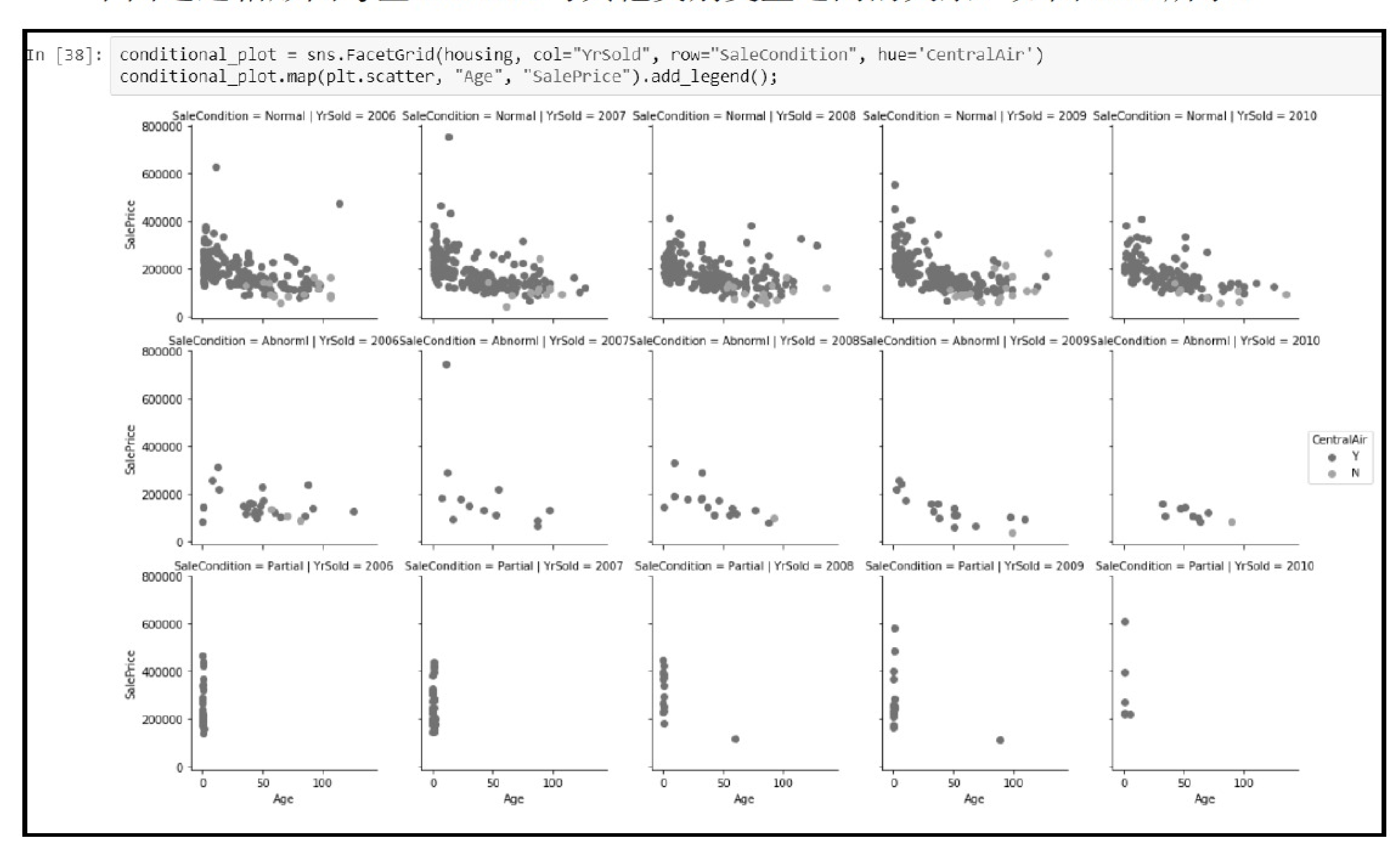


图 4.43

这里显示了变量 CentralAir（置于 x 轴）和变量 SalePrice（置于 y 轴）的箱形图。不难发现，对于未配置的 CentralAir 房屋，SalePrice 较低，房价的分布也低于配置了 CentralAir 的房屋。

除此之外，类似于散点图，还可以在一次可视化操作中展示多个箱形图。根据之前的多子图技术，可遍历每幅子图，并通过 Python 生成类别变量和 SalePrice 间的可视化结果，如图 4.44 所示。

通过观察可知，变量 MSZoning 在类别中的分布有所不同；RM 分类包含了较低售价的分布状态；在 Neighborhood 变量中，可以看到，不同的社区也会存在不同的分布状态。

图 4.45 显示了 Neighborhood 和 SalePrice 变量之间的关系。

```
In [33]: fig, ax = plt.subplots(3,3, figsize=(14,9))
         for var, subplot in zip(categorical_vars, ax.flatten()):
             sns.boxplot(x=var, y='SalePrice', data=housing, ax=subplot)

         fig.tight_layout()
```

图 4.44

```
In [34]: fig, ax = plt.subplots(figsize=(14,4))
         sns.boxplot(x='Neighborhood', y='SalePrice', data=housing, ax=ax);
```

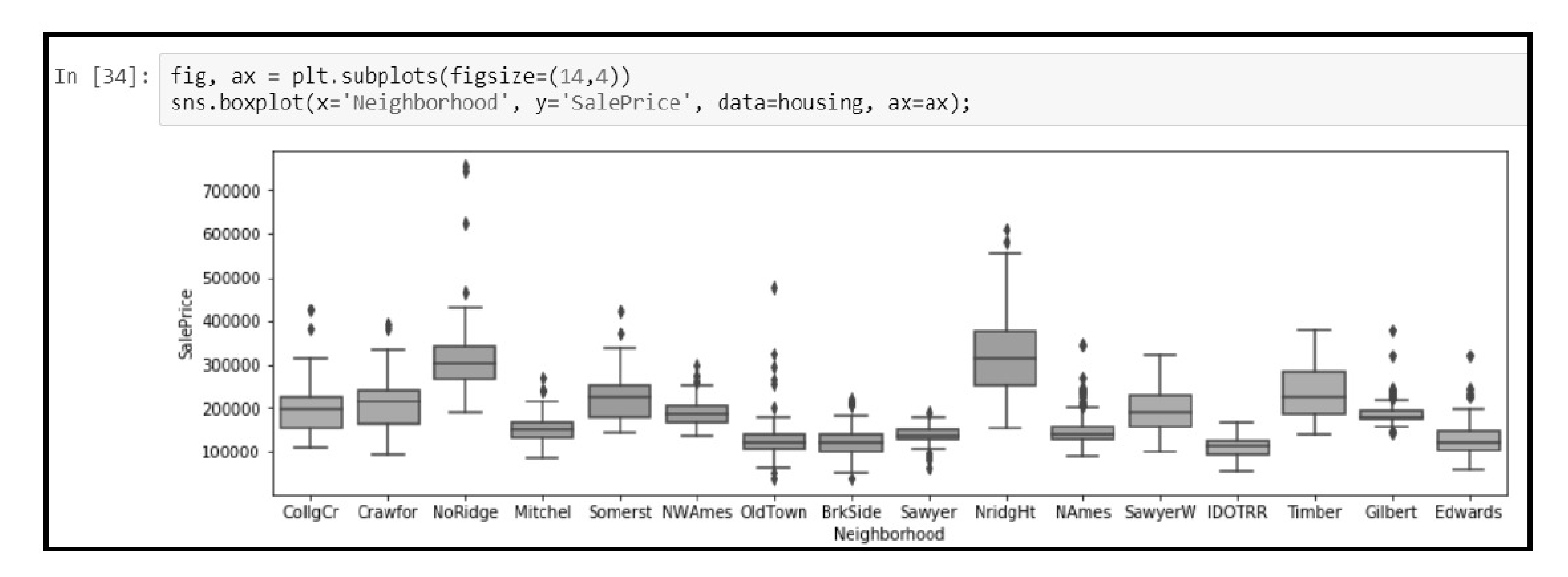


图 4.45

可以看到，不同的社区包含了不同的价格分布状态。对此，一种较好的方法是按照价格的高低以排序方式执行可视化操作。这可通过参数 order=sorted 方便地予以实现。


图 4.46 显示了根据中间价格排序 Neighborhood 后的结果。

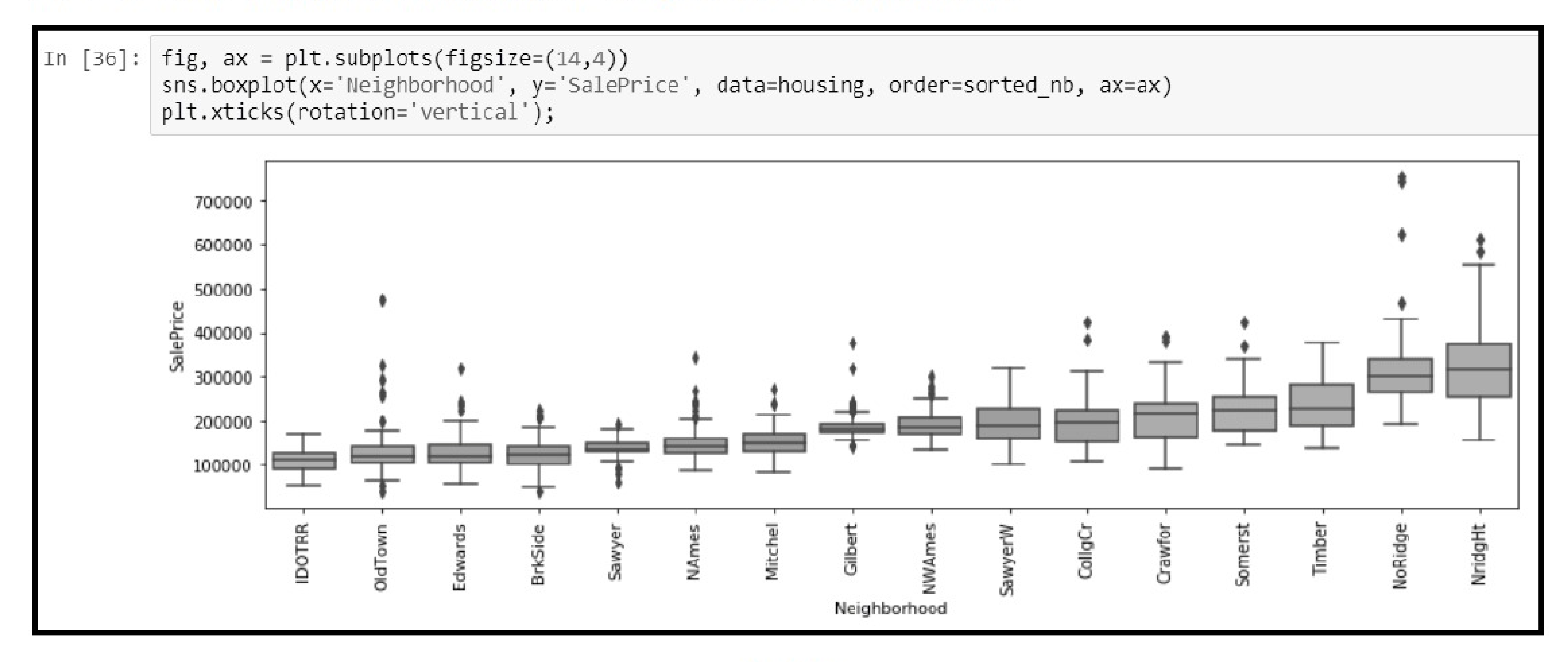


图 4.46

在根据中间价格对社区进行排序后，可以看到不同社区中价格分布状态的可视化结果。在最便宜的社区中，房屋的中间价格约为 100000 美元；而在最贵的社区中，房屋的中间价格在 300000 美元左右。

可以看到，对于某些社区来说，价格之间的差别很小。对于较小的箱形图，这表明全部价格间彼此接近；而对于某些较大的箱形图，价格分布则存在较大的偏差。因此，根据视觉显示效果，可以查看到大量的信息。

4.7.3　复杂的条件图

条件图相对复杂，并可于其中对一个变量进行条件设置。例如，如果对 Neighborhood 进行条件设置，则可利用 FacetGrid 函数生成条件图。因此，我们将对 Neighborhood 设置条件，并于随后生成 OverallQual 和 SalePrice 变量间的散点图，如图 4.47 所示。

此处，在可视化结果中的每幅散点图均在各个社区上设置了条件。如果全部社区中两个变量间具有正关系，那么就可以说，观测到的关系适用于每个社区。

如果打算可视化多个变量，可通过 FacetGrid 方法生成条件图，并传递类别变量作为行和列，同时辅以额外的类别变量并通过不同的颜色显示散点图的绘制点。根据这一新的特性，可对 Age 和 SalePrice 间的关系执行可视化操作，但此处对房屋的销售年份和 SaleCondition 变量设置条件，对应结果如图 4.48 所示。

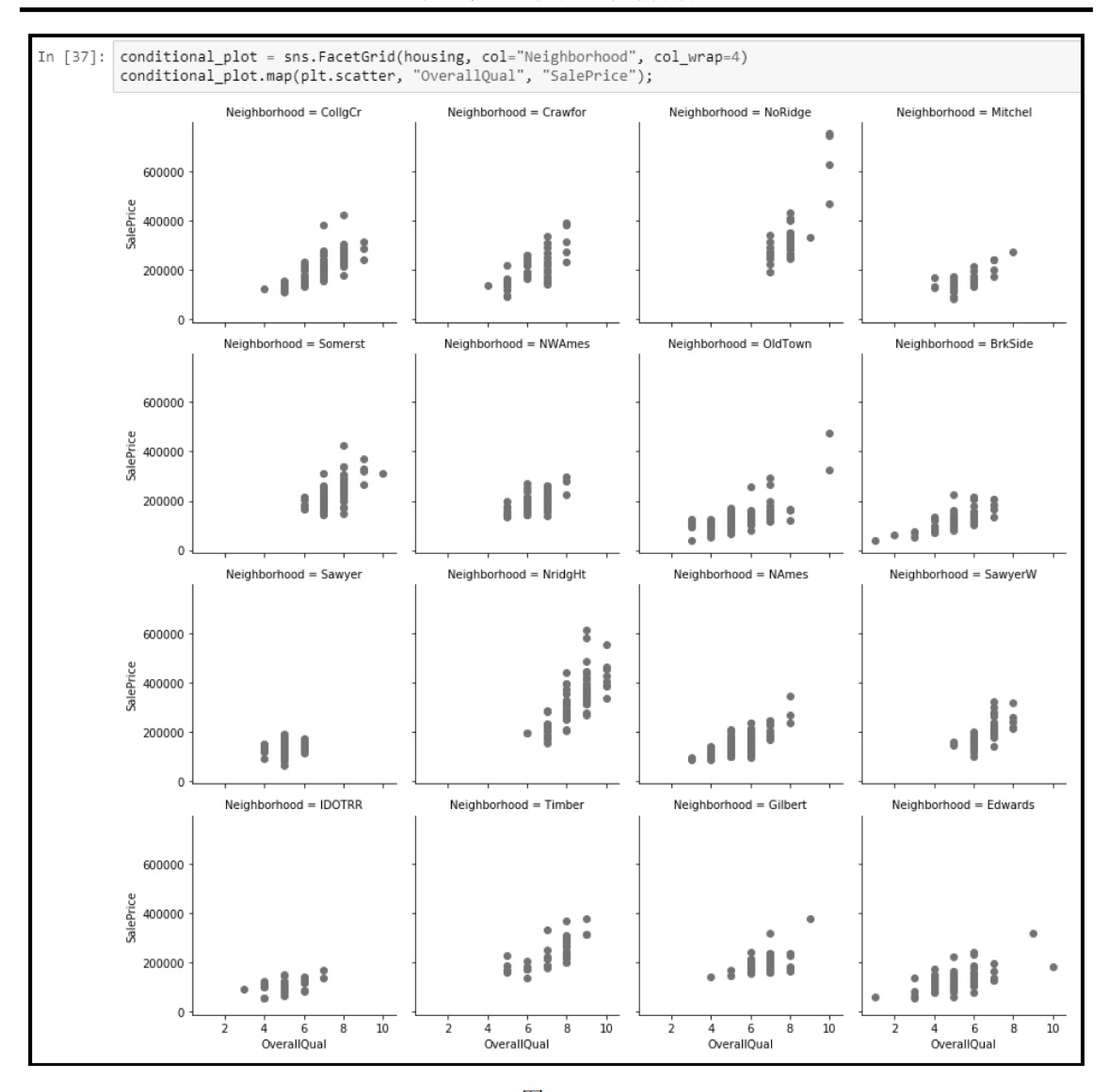


图 4.47

其中，每一列对应于每一年（2006—2010 年）；三行则分别对应于 Normal 条件、Abnormal 条件和 Partial 条件。我们还发现，Age 与 SalePrice 之间的负关系在 Normal 行的每一个子分组中都成立，并且随着年龄的增长，它趋于下降势态。深色点对应的是那些没有安装中央空调（CentralAir）的房子，因为大部分的深色点位于 50～100，因而可以判断出它们是一些老旧的房子。从图 4.48 中还可以看到，“SaleCondition 为 Partial”这一类观测结果较少；而且，对于大多数观测结果来说，房屋的年龄为 0。

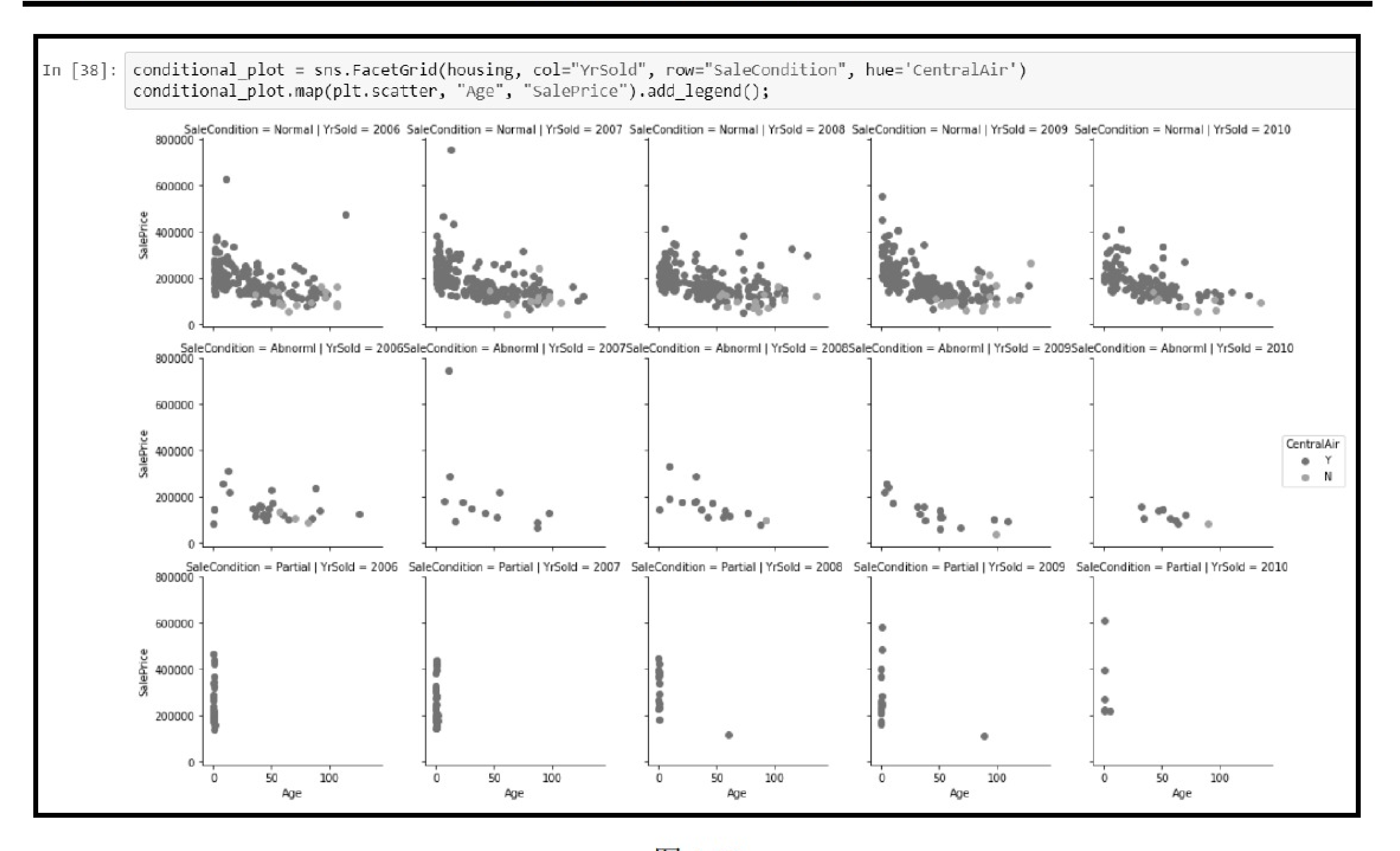


图 4.48

当前示例说明，我们可以很方便地生成复杂的可视化结果，以及如何从可视化结果中获得有价值的信息。作为数据分析人员，我们不仅应该尝试如何生成这些可视化结果，还应该从中获取一些有价值的见解。

4.8　本 章 小 结

本章讨论了 matplotlib 及其工作方式。除此之外，我们还学习了面向对象接口和 pyplot 接口之间的差异。另外，本章还介绍了一些常见的可视化操作，并探讨了如何通过 seaborn 和 pandas 执行可视化操作。最后，本章还阐述了变量间关系的可视化方式，并在真实的数据集上执行了探索性数据分析。

第 5 章将考查如何利用 Python 执行统计计算。

第 5 章　Python 统计计算

本章主要讨论 Python 科学计算库 SciPy，这可视为一个 Python 的科学计算工具箱。另外，本章还将讨论统计学子包，并以此执行多种统计计算，包括概率计算、概率分布以及置信区间。最后，我们还将在真实的数据集上执行假设测试。

本章主要涉及以下主题。

- SciPy。
- 统计学。
- 概率。
- 假设测试。
- 执行统计测试。

5.1　SciPy 简介

SciPy 是 Python 中执行科学计算的一种工具，同时也是一种基于 Python 生态环境的开源软件，涉及数学、科学计算和工程计算。SciPy 针对常见的科学计算提供了各种工具箱。对于科学计算或工程领域，相关工具可能位于 SciPy 的子包中。SciPy 的子包主要包括以下内容。

- scipy.io：该数据包提供了文件的输入/输出处理工具。
- scipy.special：该数据包提供了特定函数的处理工具。
- scipy.linalg：该数据包提供了线性代数处理工具。
- scipy.fftpack：该数据包提供了快速傅里叶转换处理工具。
- scipy.stats：该数据包提供了统计和随机数字的处理工具。
- scipy.interpolate：该数据包提供了插值处理工具。
- scipy.integrate：该数据包提供了数值积分处理工具。
- scipy.signal：该数据包提供了信号处理工具。
- scipy.ndimage：该数据包提供了图像处理工具。

5.1.1　统计子包

通过导入 stats 模块，可加载或使用统计子包，该子包包含了大量的概率分布以及不

断增长的统计函数库。由于统计是数据科学中的核心内容，因而此类软件包针对数据分析和数据科学来说，可视为一种较好的计算工具。在本章的讲解过程中，将会学习到如何利用 Python 执行一些较为常见的统计计算，并利用这些统计子包进一步理解相关的数据集（其中包含了学生饮酒信息）。

按照下列顺序，本章将利用 Scipy 中的 stats 子包执行一些常见的统计计算。

（1）引入数据集。

（2）计算置信区间。

（3）执行概率计算。

下面在 Jupyter Notebook 中考查学生饮酒量调查项目，该项目与学生的饮酒行为相关。在该项目中，所用的数据集包含了来自两所公立学校的学生信息，该数据集源自真实数据，旨在研究学生的饮酒行为与学业成绩之间的关系。此处所使用的数据集来自以下两个渠道：学校的调查报告、新生所回答的问卷。

该数据集中涵盖了 30 多个变量，这将有助于提高对问题的分析能力。下列内容显示了该数据集中的全部变量。

- ❑ school：该变量表示学生的学校（二元变量：Gabriel Pereira 表示'GP'；Mousinho da Silveira 表示为'MS'）。
- ❑ sex：该变量表示学生的性别（二元变量：'F'表示女性；'M'表示男性）。
- ❑ age：该变量表示学生的年龄（数值变量：位于 15～22）。
- ❑ address：该变量表示学生的家庭住址（二元变量：'U'表示城市；'R'表示乡村）。
- ❑ famsize：该变量表示家庭成员数（二元变量：'LE3'表示小于或等于 3 人；'GT3'表示大于 3 人）。
- ❑ Pstatus：该变量表示居住状态（二元变量：'T'表示与父母一起居住；'A'表示单独居住）。
- ❑ Medu：该变量表示母亲受教育程度（数值变量：0 表示未接受过任何教育；1 表示初等教育；2 表示 5 年级～9 年级；3 表示中等教育；4 表示高等教育）。
- ❑ Fedu：该变量表示父亲受教育程度（数值变量：0 表示未接受过任何教育；1 表示初等教育；2 表示 5 年级～9 年级；3 表示中等教育；4 表示高等教育）。
- ❑ Mjob：该变量表示母亲的工作状态（nominal：如'teacher'、'health'、'services'、'at_home'、'other'）。
- ❑ Fjob：该变量表示父亲的工作状态（nominal：如'teacher'、'health'、'services'、'at_home'、'other'）。
- ❑ reason：该变量表示选取学校的原因（nominal：如'home'、'reputation'、'course'、

'other'）。

- guardian：该变量表示学生的监护人（nominal：包括'mother'、'father'、'other'）。
- traveltime：该变量表示住址距离学校需要的时间（数值变量：1 表示小于 15 分钟；2 表示 15～30 分钟；3 表示 30 分钟～1 小时；4 表示大于 1 小时）。
- studytime：该变量表示每周的学习时间（数值变量：1 表示小于 2 小时；2 表示 2～5 小时；3 表示 5～10 小时；4 表示大于 10 小时）。
- failures：该变量表示挂课数（数值变量：1≤n<3 或 4 次）。
- schoolsup：该变量表示是否接受额外的教育支持（二元变量：yes 或 no）。
- famsup：该变量表示是否包含家教（二元变量：yes 或 no）。
- paid：该变量表示学科（Math 或 Portuguese）中是否包含额外的付费课程（二元变量：yes 或 no）。
- activities：该变量表示课外活动（二元变量：yes 或 no）。
- nursery：该变量表示是否上过幼儿园（二元变量：yes 或 no）。
- higher：该变量表示是否打算接受高等教育（二元变量：yes 或 no）。
- internet：该变量表示家中是否可访问互联网（二元变量：yes 或 no）。
- romantic：该变量表示恋爱关系状况（二元变量：yes 或 no）。
- famrel：该变量表示家庭关系状况（数值变量：1～5，分别代表较差～较好）。
- freetime：该变量表示课余时间（数值变量：1～5，分别代表较少～较多）。
- goout：该变量表示与朋友的外出时间（数值变量：1～5，分别代表较少～较多）。
- Dalc：该变量表示工作日的饮酒量（数值变量：1～5，分别代表较低～较高）。
- Walc：该变量表示周末的饮酒量（数值变量：1～5，分别代表较低～较高）。
- health：该变量表示当前健康状况（数值变量：1～5，分别代表较差～较好）。
- absences：该变量表示缺课状况（数值变量：0～93）。

年级与课程科目的关系如下。

- G1：代表第一学期成绩（数值变量：0～20）。
- G2：代表第二学期成绩（数值变量：0～20）。
- G3：代表期末考试成绩（数值变量：0～20，同时也是输出结果）。

本节主要关注以下 3 个变量。

- ac1：从当前数据中定义该变量，以存储饮酒消耗量。
- G3：该变量用于存储课程的期末考试成绩。
- gender：该变量用于存储学生的性别。

下面加载数据集，如图 5.1 所示。

```
In [5]: student.head()
Out[5]:
```

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Wal
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	

5 rows × 33 columns

图 5.1

首先设置一个酒精消耗量变量，并于其中定义一个名为 alcohol_index 的中间变量，表示学生饮酒量（平时和周末）的加权平均值，如下所示。

```
student.rename(columns={'sex':'gender'}, inplace=True)
student['alcohol index'] = (5*student['Dalc'] + 2*student['Walc'])/7
# Alcohol consumption level
student['acl'] = student['alcohol_index'] < = 2
student['acl'] = student['acl'].map({True: 'Low', False: 'High'})
```

随后通过 alcohol_index 变量将学生分为几组。如果 alcohol_index 小于或等于 2，该组的酒精消耗量较低；如果 alcohol_index 大于 2，则该组酒精消耗量较高。

5.1.2　置信区间

本节将首先计算置信区间。其中，要计算置信区间的第一个变量是期末成绩和我们感兴趣的统计结果，也就是均值。此处查看数据集中的 sample_size，且包含了 649 个观测结果。考虑到采样大小大于 30，因而可通过中心极限定理计算置信区间，如图 5.2 所示。

```
In [7]: sample_size = student.shape[0]
        print(sample_size)

        649
```

图 5.2

据此，可通过正态分布针对某个变量的均值计算置信区间。对此，仅需使用到以下 3 个数值。

- ❑ 变量的采样均值。
- ❑ 公式的标准误差。
- ❑ 置信度。

当前已知晓如何计算采样均值，如图 5.3 所示。可以看出，期末成绩对应的变量为 G3，其输出结果为 11.906009244992296。

```
In [8]: sample_mean_grade = student['G3'].mean()
        sample_mean_grade

Out[8]: 11.906009244992296
```

图 5.3

在图 5.4 中，可以看到计算采样均值的具体应用，同时还将计算标准误差。至此，我们得到了所需的前两个数字。当计算置信区间时，还将使用到 stats 子包中的 norm 对象。norm 对象定义了一个 interval()方法，并接收 3 个输入数据，即我们所需要的 3 个数值。

```
In [13]: std_error_grades = student['G3'].std()/sqrt(sample_size)

In [14]: stats.norm.interval(0.95, loc=sample_mean_grade, scale=std_error_grades)
Out[14]: (11.65745768566587, 12.154560804318722)
```

图 5.4

其中，第一个数值为当前区间的置信度，即 0.95。loc 参数表示变量的 sample_mean；scale 参数表示刚刚计算的 std_error。可以看到，针对期末成绩的均值，95%的置信区间位于 11.65745768566587～12.154560804318722。现在我们知道了如何计算均值、标准误差和范数的置信区间，下面考查一个例子，看看如何计算一个比例的置信区间。

这里将使用刚刚创建的变量，即酒精的消费水平，并查看酒精消费水平较高的学生比例的置信区间。计算过程也较为类似，也就是说，需要使用到以下 3 个数字。

- ❑ 样本比例。
- ❑ 样本标准偏差。
- ❑ 置信度。

针对现有比例，标准误差的计算公式为

$$SE = \sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

其中，$\hat{p}$ 表示为样本比例。下面考查酒精消费较高的学生的样本比例。通过图 5.5

可以看到，该值约为 0.255778。

```
In [11]: student['acl'].value_counts(normalize=True)

Out[11]: Low     0.744222
         High    0.255778
         Name: acl, dtype: float64
```

图 5.5

借助于 high_prop 变量，下列代码将使用该公式计算标准误差和置信区间。

```
high_prop = student['acl'].value_counts(normalize=True)['High']
std_error_prop = sqrt(high_prop*(1-high_prop)/sample_size)
```

但这一次，将针对学生比例计算 98%的置信区间（在酒精消费较高的人群中），如图 5.6 所示。

```
In [13]: stats.norm.interval(0.98, loc=high_prop, scale=std_error_prop)

Out[13]: (0.21593666225148048, 0.2956195781183193)
```

图 5.6

输出值为 0.21593666225148048～0.2956195781183193。因此，对于当前学生比例，0.25 似乎是一个较好的猜测值。

记住，这些仅是样本中的计算；同时，我们使用推论统计对相关人群进行判断。

5.1.3　概率计算

假设总体概率的对应值为 0.25，下面通过该值进行统计计算。

stats 数据包中存在多种概率分布，进而可据此进行模拟、计算随机变量并执行概率计算。下面考查相关示例并对某些问题予以解答。

假设发现一名酒精消费水平较高的学生的概率是 0.25，如果我们有一个 10 人的班级，那么，发现 5 名酒精消费水平较高的学生的概率是多少?

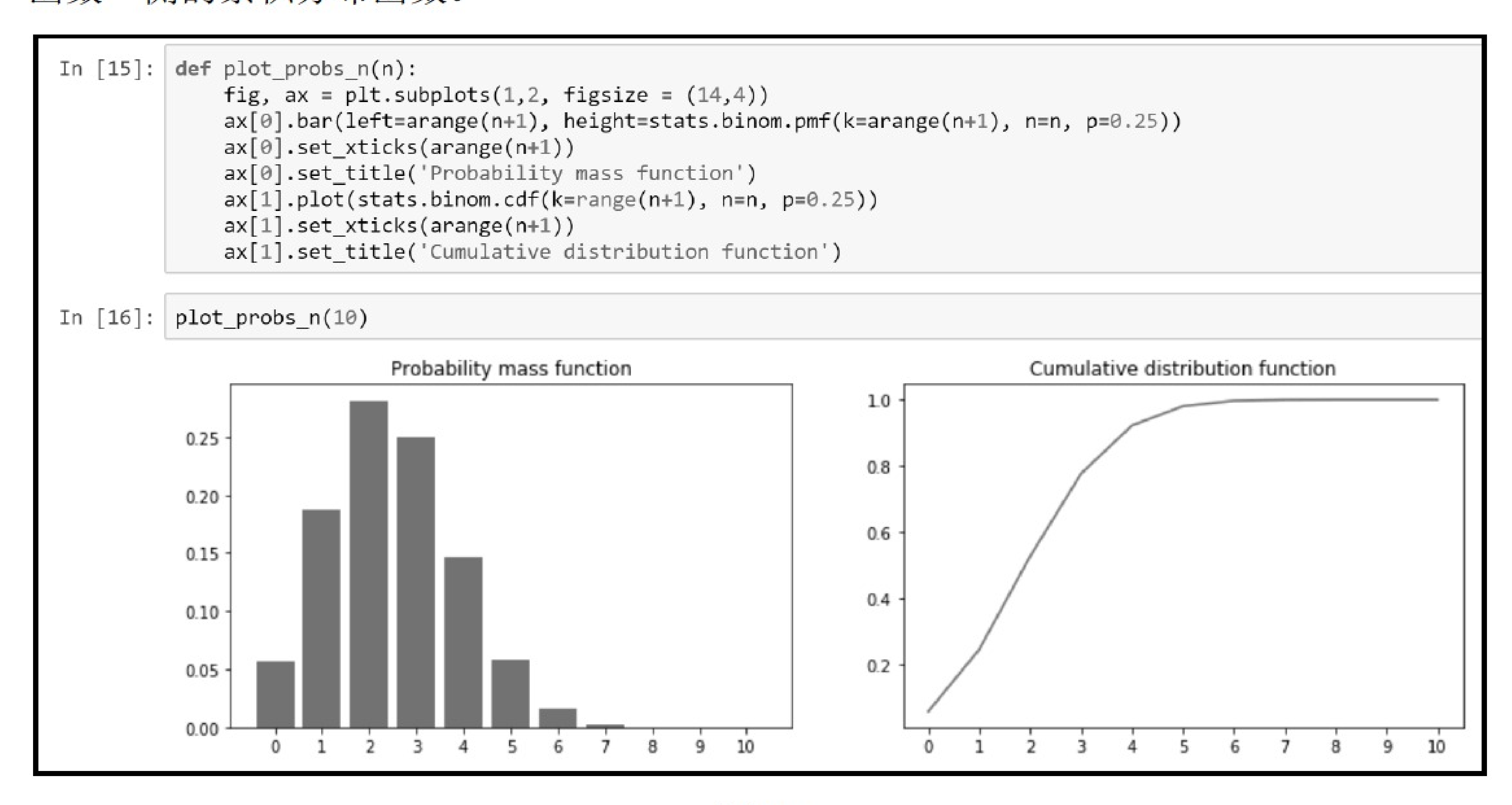
对此，可通过二项分布计算这一概率。另外，stats 数据包中定义了 binom 对象，其中，定义了一个名为概率质量函数的方法，即 pmf，我们需要针对二项分布提供 pmf 所需的参数。当前，我们关注的是在一组 10 名学生中计算 5 名酒精消费水平较高的学生的概率，且总体概率是 0.25。具体的计算过程和概率结果如图 5.7 所示。

```
In [14]: stats.binom.pmf(k=5, n=10, p=0.25)

Out[14]: 0.058399200439453194
```

图 5.7

可以看到，当前事件的概率为 0.058399200439453194。当然，也可同时计算多个概率。在图 5.8 中，一方面，获取特定学生数量的概率位于 x 轴上，而柱状图则表示在 10 名学生组成的班级中，获取酒精消费水平较高的学生的概率；另一方面，一般难以找到这样的班级，其全部学生的酒精消费水平均较高。另外，在图 5.8 中，还可查看概率质量函数一侧的累积分布函数。



```
In [15]: def plot_probs_n(n):
             fig, ax = plt.subplots(1,2, figsize = (14,4))
             ax[0].bar(left=arange(n+1), height=stats.binom.pmf(k=arange(n+1), n=n, p=0.25))
             ax[0].set_xticks(arange(n+1))
             ax[0].set_title('Probability mass function')
             ax[1].plot(stats.binom.cdf(k=range(n+1), n=n, p=0.25))
             ax[1].set_xticks(arange(n+1))
             ax[1].set_title('Cumulative distribution function')

In [16]: plot_probs_n(10)
```


图 5.8

因此，可使用 binom 分布中的 cdf()方法计算累积分布函数。

5.2　假设测试

本节将执行假设测试，以回答“饮酒量是否会对学生的成绩带来影响”这一类问题。本节主要涉及以下主题。

- 零假设测试框架。
- 对总体方差的相等性进行测试。
- 对总体均值的相等性进行 t 测试。

随后，我们将根据所得结果回答上述问题。对此，将采用 Jupyter Notebook 快速浏览一下相关步骤，这也是采用假设测试回答某个问题时的常见作法，具体如下。

（1）此处将建立两个相互竞争的假设，即零假设和备择假设（alternative hypothesis）。所回答的问题类型决定了具体采用哪一种假设。

（2）这里将提前设置一个显著性水平（significance level），称作 Alpha。通常，研究人员或数据分析师一般会选取 5%或 0.05；其他常见的选择值还包括 0.01 或 0.1（即 10%）。

（3）在 p 值中计算测试统计结果，并于随后将这一 p 值与 Alpha 值进行比较。

（4）对于是否采用零假设，可根据 p 值和显著性水平的比较结果，选择拒绝或接受零假设。

（5）最终，在对应结果与最初问题之间，需要给出一个总体的结论。

5.3　执行统计测试

在进行统计测试之前，下面首先考查一些统计函数，进而可通过 scipy 数据包进行测试，具体如下。

- kurtosistest(a[, axis, nan_policy])：该数据包用于测试数据集是否包含正太峰度。
- normaltest(a[, axis, nan_policy])：该数据包用于测试样本是否不同于正太分布。
- skewtest(a[, axis, nan_policy])：该数据包用于测试斜度是否不同于正态分布。
- pearsonr(x, y)：该数据包用于计算皮尔森相关系数，以及非相关测试的 p 值。
- ttest_1samp(a, popmean[, axis, nan_policy])：该数据包用于计算一组分值的平均值的 t 测试。
- ttest_ind(a, b[, axis, equal_var, nan_policy])：该数据包用于计算两个独立分值样本均值的 t 测试。
- ttest_ind_from_stats(mean1, std1, nobs1, ...)：该数据包用于对描述性统计中两个独立样本的均值进行 t 测试。
- ttest_rel(a, b[, axis, nan_policy])：该数据包用于计算两个相关分值样本（a 和 b）上的 t 测试。
- kstest(rvs, cdf[, args, N, alternative, mode])：该数据包用于执行 Kolmogorov Smirnov 拟合优度测试。
- chisquare(f_obs[, f_exp, ddof, axis])：该数据包用于计算单向卡方测试。
- ansari(x, y)：该数据包用于执行等尺度参数的 Ansari-Bradley 测试。
- bartlett(*args)：该数据包用于执行 Bartlett 的等方差测试。

- levene(args, *kwds)：该数据包用于执行 Levene 的等方差测试。
- shapiro(x[, a, reta])：该数据包用于执行 Shapiro-Wilk 正态测试。
- anderson(x[, dist])：针对来自特定分布的数据，该数据包用于执行 Anderson-Darling 测试。
- anderson_ksamp(samples[, midrank])：该数据包用于执行 Anderson-Darlingk k 采样测试。

上述内容展示了一些较为流行的统计测试。如果读者访问 scipy.stats 子包中的文档（对应网址为 https://docs.scipy.org），还会发现更多的专项测试。

下面通过某项测试回答以下问题：两组学生的总体方差是否相等？其中，一组学生的饮酒量较低，而另一组学生的饮酒量较高。鉴于该问题与方差相关，因而可采用 Bartlett 测试，其零假设表明，总体方差是相等的，如图 5.9 所示。

```
In [18]: grades_low_acl = student['G3'][student['acl']=='Low']
         grades_high_acl = student['G3'][student['acl']=='High']
         stats.bartlett(grades_low_acl, grades_high_acl)

Out[18]: BartlettResult(statistic=1.1025085913378183, pvalue=0.29371623181175127)
```

图 5.9

图 5.9 中显示了样本中的方差，且看起来彼此接近。下面通过一项正规的测试以对总体方差问题得出某些结论。图 5.9 中分别定义了表示 Low 分组中学生成绩的向量、High 分组学生成绩向量，以及一个 Bartlett 函数。相应地，传递至该函数的唯一内容即是刚刚创建的两个向量。

此处须关注 pvalue，对应值为 0.29371623181175127，大于之前的显著性水平，即 5% 或 0.05。因此，根据该测试，我们不能排除等方差的零假设。相应地，可假设这两组分数源自具有相同方差的一个整体。

在下一个测试中，将尝试回答下列问题：酒精的消耗量是否会对学业带来影响？当查看数据或样本数据的可视化结果时，可以看到两组学生的成绩，或者说平均成绩，存在着非常明显的差异。在图 5.10 中，与酒精消费量较大的一组学生相比，饮酒量较低的一组学生其成绩平均值较高。

下面尝试构建两种假设。针对当前问题，零假设可描述为：两组期末成绩的总体均值是相等的；备择假设则表明，期末成绩的总体均值是不同的。在进行任何测试之前，需要记住，全部测试均包含了某些假设条件，当前即定义了 3 个测试假设条件，并验证这 3 个假设在当前情况下是成立的，包括第 3 个假设条件，即方差相等——我们刚刚利用之前的测试证明了这一点。

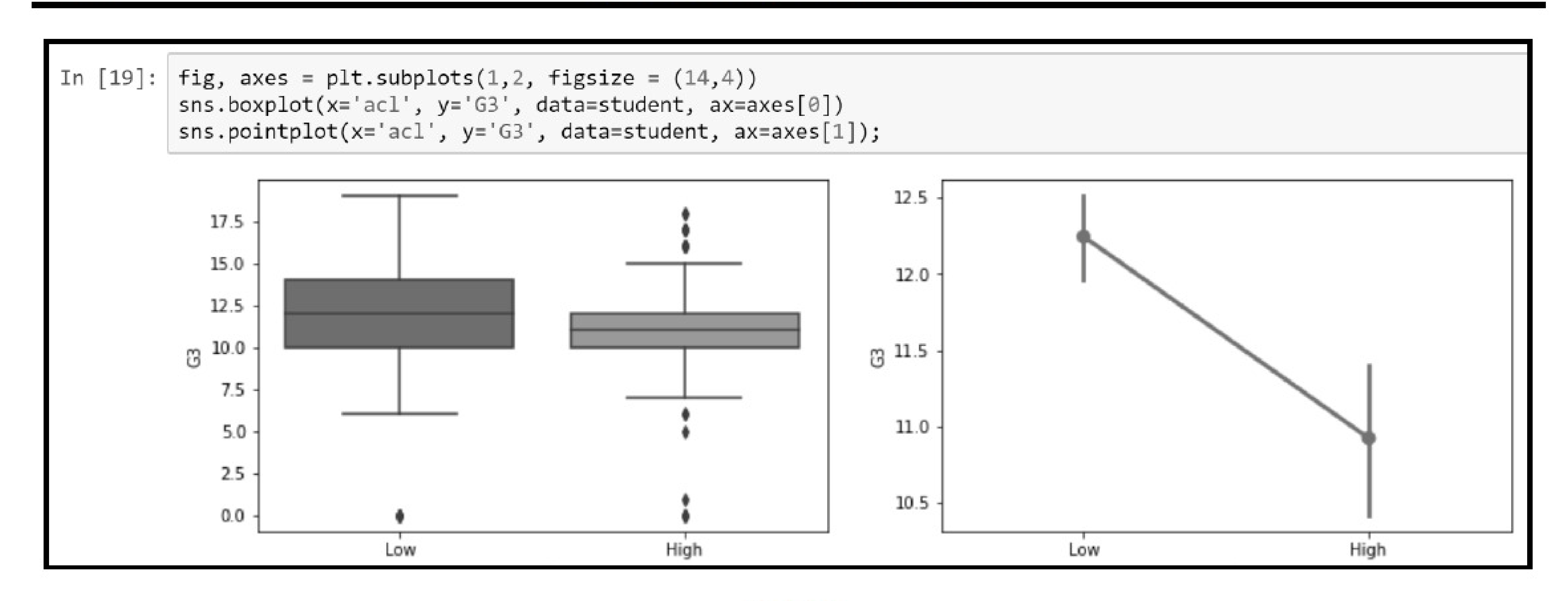


图 5.10

当执行 t 测试时，可使用 stats 数据包中的 ttest_ind 函数。其间，可传递包含两组学生成绩的两个向量，并设置一个附加参数以表示方差是相等的。当执行该测试时，可回顾一下之前讨论的统计结果和 p 值。在图 5.11 中，可以看到 p 值较低，即 4.6036088303692686e-06。

```
In [20]: stats.ttest_ind(grades_low_acl, grades_high_acl, equal_var=True)

Out[20]: Ttest_indResult(statistic=4.621320706949354, pvalue=4.6036088303692686e-06)
```

图 5.11

总之，这是支持备择假设的一个有力的证据。也就是说，当比较两组内容时，成绩的总体均值实际上是不同的。因此，最终结论如下：当分析两组成绩时，二者间在统计学意义上存在着显著的差异。由于高酒精摄入量的一组学生其平均值低于另一组学生的平均值，结果表明，酒精摄入量对学生的学业存在着负面影响。

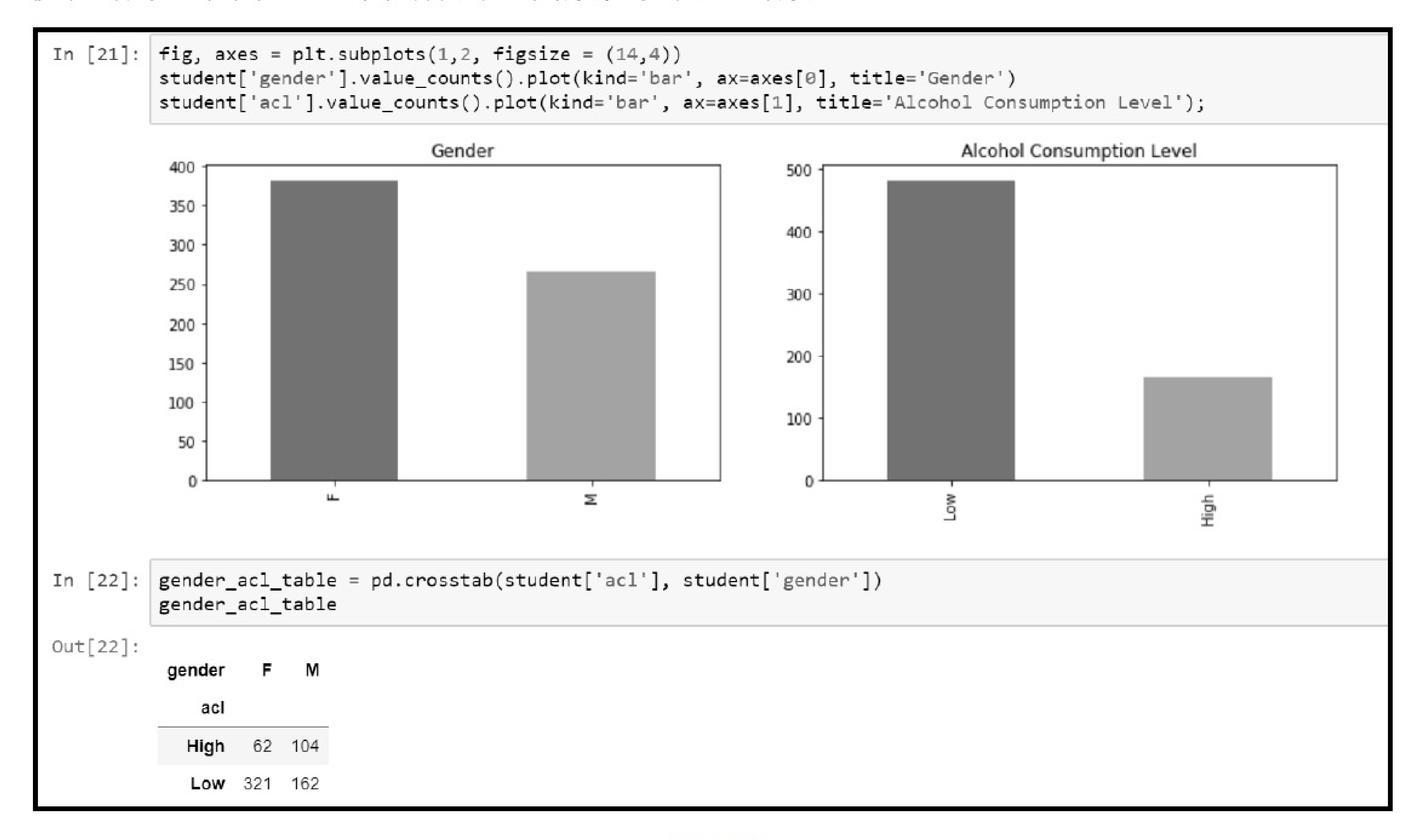
下面通过假设测试回答以下问题：男学生的饮酒量是否高于女学生？这一问题涉及以下两个主题。

- 对关联表执行卡方测试。
- 对两个关联表的关系执行可视化操作。

对此，可尝试在 Jupyter Notebook 进行操作，如图 5.12 所示。

在图 5.12 中可以看到，女性的数量多于男性，且低饮酒量的学生占据了大多数。对于关联表（也称作交叉表），存在多种方法可对其进行计算。这里将采用 pandas 中的 crosstab 函数并传递两个 Series，输出结果表示为每个分类的计数值。当考查这两个类别变量时，还可能涉及 4 种分类。因此，分别有 62 位女性和 104 位男性，其酒精摄入量较高。另外，我们还统计了低酒精消耗量的女性和男性数量。图 5.13 显示了此类数值的可

视化结果，其中涵盖了具体的比例数值以及绝对数量。



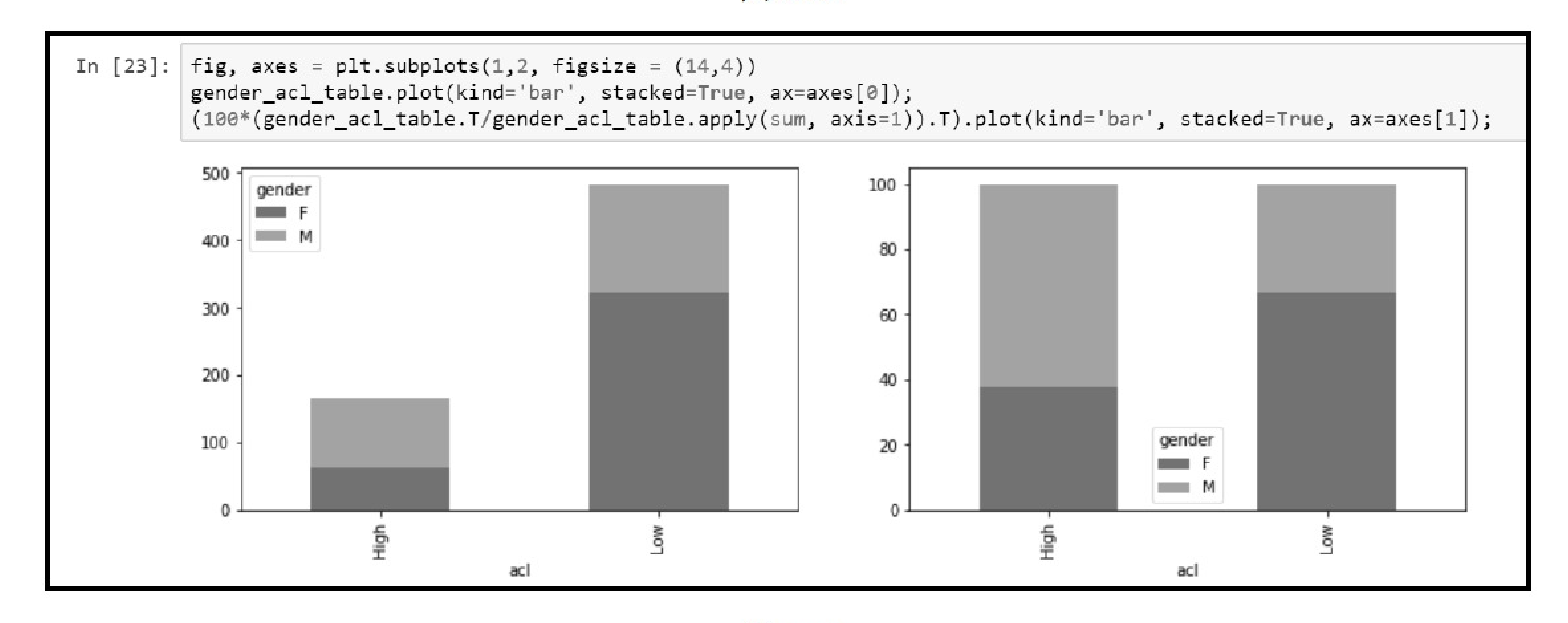
In [21]:
```
fig, axes = plt.subplots(1,2, figsize = (14,4))
student['gender'].value_counts().plot(kind='bar', ax=axes[0], title='Gender')
student['acl'].value_counts().plot(kind='bar', ax=axes[1], title='Alcohol Consumption Level');
```


In [22]:
```
gender_acl_table = pd.crosstab(student['acl'], student['gender'])
gender_acl_table
```

Out[22]:

gender	F	M
acl		
High	62	104
Low	321	162

图 5.12



In [23]:
```
fig, axes = plt.subplots(1,2, figsize = (14,4))
gender_acl_table.plot(kind='bar', stacked=True, ax=axes[0]);
(100*(gender_acl_table.T/gender_acl_table.apply(sum, axis=1)).T).plot(kind='bar', stacked=True, ax=axes[1]);
```


图 5.13

在图 5.13 中可以看到，一方面，Low 分组中的女性比例大于男性；另一方面，对于酒精摄入量较大的一组，男性占据了较大的比例。因此，两个变量间似乎包含了某种关系，样本中显然也涵盖了某种关系。这里的问题是，这种关系适用于总体吗？为了回答这一问题，我们需要执行假设测试，如图 5.14 所示。

```
In [24]: chi_stat, p_value, dof, expected = stats.chi2_contingency(gender_acl_table)

In [25]: p_value
Out[25]: 8.72933011769437e-11

In [26]: expected_table = pd.DataFrame(expected, index=['High','Low'], columns=['F','M'])
         expected_table
Out[26]:
```

	F	M
High	97.96302	68.03698
Low	285.03698	197.96302

图 5.14

从图 5.14 可以看到，当前示例中所用的 chi2_contingency 函数源自 stats 数据包，并传递了之前创建的关联表。该函数将生成多个对象，如 chi-square statistic、p_value、自由度（dof）以及零假设下包含期望值的一个表。在当前示例中，零假设可描述为：总体中两个变量间不存在任何关系。当前示例执行测试后，对应结果如图 5.15 所示。

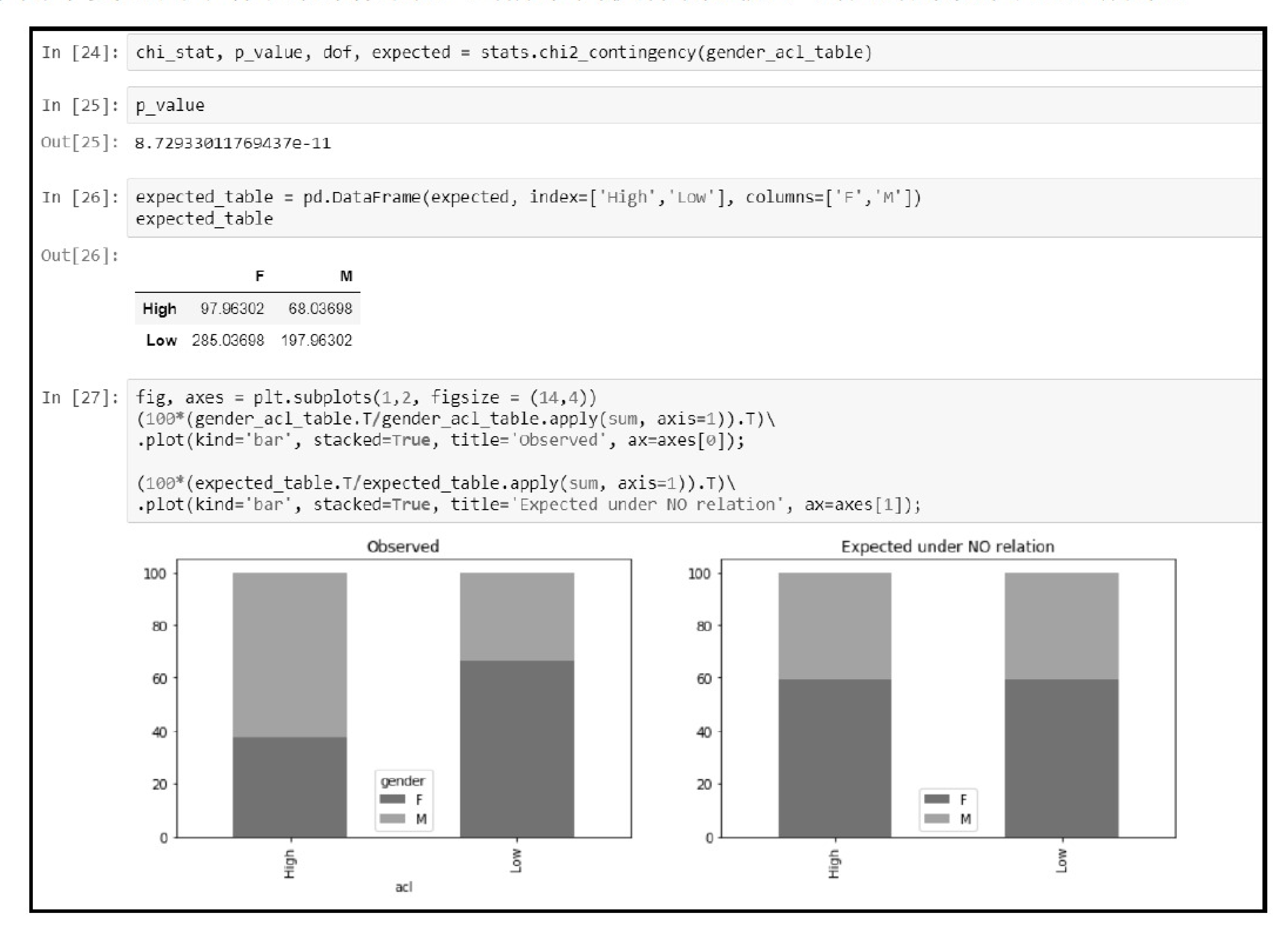


图 5.15

对于图 5.15 中的 p_value 对象，可以看到该值较低，即 8.72933011769437e-11，这也为备择假设提供了较为明显的证据。在当前示例中，备择假设是指两个变量间实际上存在某种关系。另一个令人感兴趣的对象则是 expected。如果零假设成立，这将是我们观察到的期望频率。

需要记住的是，上述示例测试的零假设可描述为：两个变量间不存在某种关系。因此，如果总体中不存在任何关系，则可对 In [32]执行可视化操作，进而观察数据的状态并进行适当比较。相应地，这可视作一个额外的证据，进而展示了男性学生的饮酒习惯与女性学生的饮酒习惯之间的差异。不难发现，男性学生的饮酒量较高。

5.4　本 章 小 结

本章讨论了 SciPy，并于随后对各种子包进行了简要的介绍。除此之外，我们还学习了如何利用 scipy.stats 数据包执行卡方测试，进而执行其他的一些计算，如置信区间、概率计算以及其他的统计测试类型。

第 6 章将讨论预测分析模型、机器学习和 scikit-learn 库等内容。

第 6 章　预测分析模型

本章主要涉及以下主题。

- ❑ 预测分析。
- ❑ 机器学习。
- ❑ scikit-learn 库。
- ❑ 如何构建分类和回归模型。

6.1　预测分析和机器学习

本节将介绍预测分析的基础知识，并深入了解基于预测分析的机器学习方案，以及各种类型的机器学习模型。在本节的最后还将会介绍监督学习模型的组成部分。

提示：

在当前上下文环境中，预测是指对未知或尚未观察到的结果进行猜测，而与未来无关。

存在多种方式可对事物进行预测。例如，通过直觉或者请教专家进行判断，这些都是较为传统的商业惯例。最近，较为成功的方法就是预测分析。

近些年来，预测分析的应用得到了显著的发展，主要涉及以下两个原因。

- ❑ 科技的发展为我们提供了实现这种分析的手段。
- ❑ 预测分析具有非常有效、准确的特征，并且工作良好。

预测分析是将数据与数学、统计学和计算机科学技术相结合，进而对未知事件进行预测。预测分析的目标是对未知事件可能发生的情况进行良好的评估。

那么，如何执行预测分析。对此，存在多种解决方案。近期，机器学习表现得较为突出，为了进一步理解其工作方式，下面首先对机器学习展开讨论。

机器学习是计算机科学的一个子领域，并可以简单地描述为：无须显式地编程即可赋予计算机某种学习能力。机器学习领域已经发展出许多方法，以使计算机使用数据执行某些任务。这些方法在预测分析方面非常成功。当然，机器学习也存在一些缺点，理解机器学习可能会遇到的问题是十分重要的。我们可以把此类学习问题分成几个大类。为了简化这一问题，可以考虑两组学习模式，即监督学习和无监督学习。本节主要讨论

监督学习，以及构建监督机器学习所需的各种元素。

在考虑创建机器学习模型时，首先应想到的问题是：如何知道应该使用监督式学习？理解这一点的关键在于获取需要预测的目标变量。监督式学习包含了与特定概念/主题相关的样本或观测结果。其中，每个观测结果都具有不同的特征，这些特征通常称作属性或目标变量，同时需要对这些目标变量进行预测。

下面考查包含多行学生数据的数据集。图 6.1 分别显示了性别、年龄、住址等特征信息。

	school	sex	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	famrel	freetime	goout	Dalc	Walc	health	absences	G1	G2	G3
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	3	4	1	1	3	4	0	11	11
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	5	3	3	1	1	3	2	9	11	11
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	4	3	2	2	3	3	6	12	13	12
3	GP	F	15	U	GT3	T	4	2	health	services	...	3	2	2	1	1	5	0	14	14	14
4	GP	F	16	U	GT3	T	3	3	other	other	...	4	3	2	1	2	5	0	11	13	13

5 rows × 33 columns

图 6.1

如果将上述特征之一定义为需要预测的目标变量，随后即可使用监督式学习方案。

监督式学习主要考查两方面的内容，即回归和分类。二者间的差别十分简单，一方面，当目标变量在本质上是类别变量时，即会进行分类，这方面的一些例子包括酒精消费水平（低或高）或信用卡交易的类型；另一方面，如果目标变量是一个数值变量，则会涉及回归问题，数值变量的一些相关示例包括房屋或股票的价格，或者一个月内售出单元的数量。针对每一类问题，分别存在多种模型可对其进行修正。

综上所述，首先，可将本节中的模型视为一个黑盒，这意味着我们不会解释与内部工作方式相关的细节内容。考虑到这是对预测分析的整体介绍，因而接下来将关注如何构建预测模型的全局内容。这一原则同样适用于机器学习中的大多数概念。

6.2　理解 scikit-learn 库

本节主要介绍 scikit-learn 库，并以此实现简单的预测模型。对此，需要深入理解 scikit-learn，以及如何向 Jupyter Notebook 加载 iris 数据集。随后，将详细讨论如何利用 scikit-learn 构建机器学习模型，并在此基础上实现简单的预测模型。

scikit-learn 是针对机器学习的一个较为流行的 Python 库，同时面向数据建模和数据分析提供了高效的 API 工具。scikit-learn 构建于 NumPy、SciPy 和 Matplotlib 之上。图 6.2 显示了 Jupyter Notebook 中的操作结果。

```
In [1]: from sklearn import datasets

In [2]: import pandas as pd
        import numpy as np

In [3]: iris = datasets.load_iris()
        iris_features = iris.data
        iris_target = iris.target

In [4]: iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
        iris_df['target'] = iris.target_names[iris.target]
        iris_df.head()

Out[4]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

图 6.2

此处并未导入全体库，而是有针对性地导入了某些库。另外，我们需要导入 datasets 对象，进而可加载 scikit-learn 提供的所有数据集。

为了更好地理解上述概念，这里将采用 iris 作为示例。图 6.3 中显示了 iris 数据集中的一些数据行。

```
In [4]: iris_df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
        iris_df['target'] = iris.target_names[iris.target]
        iris_df.head()

Out[4]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

图 6.3

该数据集包含了与花朵相关的 150 个观测结果，且每朵花对应于 4 个测量数据，分别表示为 sepal length、sepal width、petal length、petal width。除此之外，此处还定义了花的种类，并将其用作目标变量。

数据集中提到的物种包括 setosa、versicolor 和 virginica。如果使用数据集中的测量值

来预测花朵的种类，这将是一个类别变量。也就是说，我们正在执行一项分类任务。scikit-learn 中实现的主要 API 则是估计函数。估计函数对象中包含了数据学习所用的模型。

在 Jupyter Notebook 中，首先需要导入估算函数（常称作模型）。下列代码用于导入这一分类器。

```
from sklearn.neighbors import KNeighborsClassifier
```

这里所用的模型称作 KNeighbors 模型，并从 scikit-learn 中予以导入。随后，将生成该对象实例，即 flower_classifier，对应代码如下。

```
flower_classifier = KNeighborsClassifier(n_neighbors=3)
```

这里，我们向当前对象提供了超参数。如前所述，此类对象可视作黑盒。因此，我们并不会深入讨论与特定数值或变量相关的一些问题。

然后，可利用这些数据来训练估计函数。为此，此处使用 flower_classifier 对象的 fit 方法，这有助于传递相关特性和目标。这意味着当前已经使用了特性来识别目标。图 6.4 所示代码及其输出用于训练估算函数。

```
In [10]: flower_classifier.fit(X=iris_features, y=iris_target)

Out[10]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                    metric_params=None, n_jobs=1, n_neighbors=3, p=2,
                    weights='uniform')
```

图 6.4

当前，对应模型可进行评估操作，这也是我们无法查看到的内容。假设评估结果令人满意，则可利用该模型进行预测。图 6.5 显示了用于预测花朵种类的相关特征。

注意：

这里，所用数组须为二位 NumPy 数组。

数组的输出结果为 0 时，对应物种将被分类为 setosa；另外，数值 1 和 2 则将物种分类为 versicolor 和 virginica。

当预测花卉的物种时，可使用分类器对象，并于随后传递 new_flower1，对应的输出结果如图 6.6 所示。

根据所接收到的输出结果，标记为 0 的物种为 setosa。通过类似的方式，还可对第二朵花的物种进行预测。针对不同的测量行为，可持续执行相同的操作。

通过创建包含各种花朵值的 *n* 个 NumPy 数组，还可查看累积预测结果，我们可将其称作 new_flowers 函数。图 6.7 显示了所定义的 predictions 函数。

```
In [11]: # The features must be two-dimensional array
         new_flower1 = np.array([[5.1, 3.0, 1.1, 0.5]])
         new_flower2 = np.array([[6.0, 2.9, 4.5, 1.1]])

         0 == > setosa

         1 == > versicolor

         2 == > virginica

In [12]: flower_classifier.predict(new_flower1)

Out[12]: array([0])

In [13]: flower_classifier.predict(new_flower2)

Out[13]: array([1])

In [14]: new_flowers = np.array([[5.1, 3.0, 1.1, 0.5],[6.0, 2.9, 4.5, 1.1]])
         predictions = flower_classifier.predict(new_flowers)
         predictions

Out[14]: array([0, 1])
```

图 6.5

```
In [12]: flower_classifier.predict(new_flower1)

Out[12]: array([0])
```

图 6.6

```
In [14]: new_flowers = np.array([[5.1, 3.0, 1.1, 0.5],[6.0, 2.9, 4.5, 1.1]])
         predictions = flower_classifier.predict(new_flowers)
         predictions

Out[14]: array([0, 1])
```

图 6.7

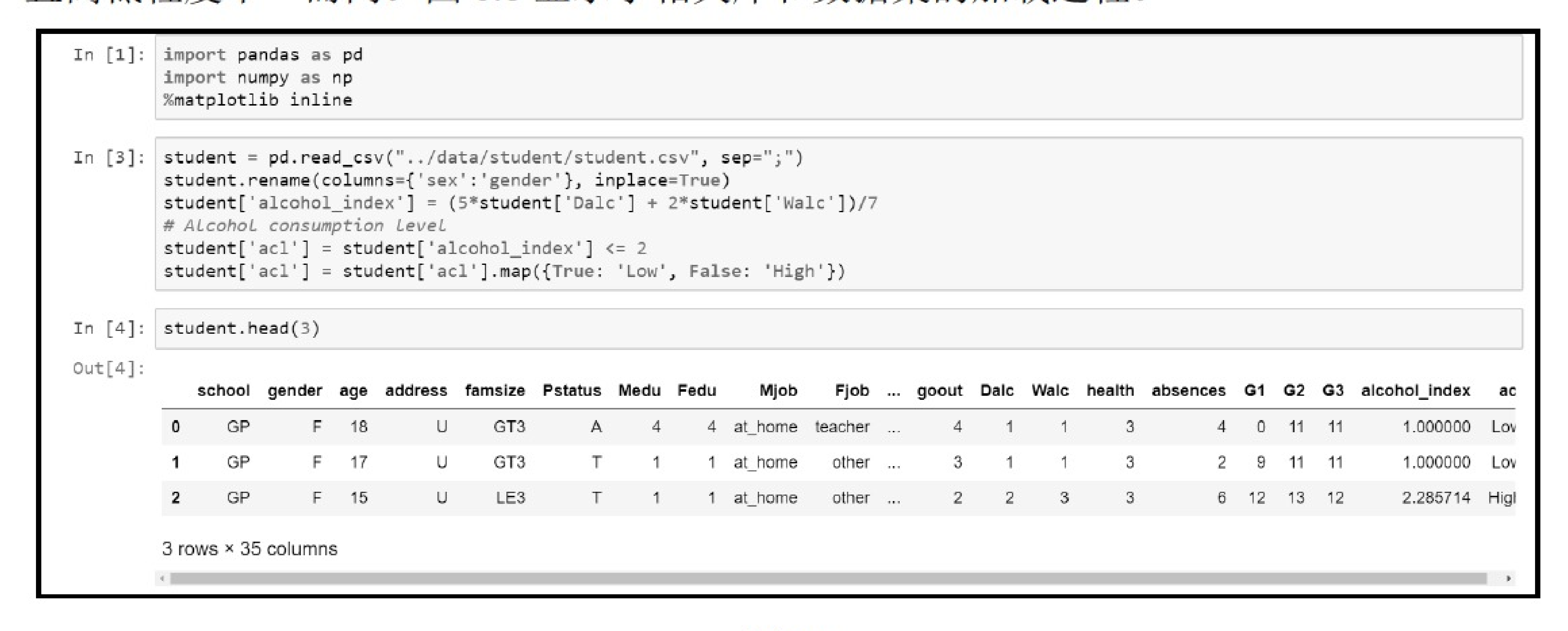
通过观察可知，第一个数值对应于第一朵花，并将其分类为 setosa；第二个数值对应于第二朵花，并将其分类为 versicolor。

6.3　使用 scikit-learn 构建回归模型

6.2 节讨论了基于 scikit-learn 的分类模型示例。本节将对随机森林模型进行训练，并以此进行预测。此外，本节还将构建一个分类模型，并作为当前环境下的目标变量，这将是一个描述青少年饮酒习惯的分类值。

对此，首先需要加载之前提供的学生数据集，并于随后训练逻辑回归模型，进而考查如何在较为基础的水平之上评估该分类模型。

在开始阶段，可加载相关库，导入学生数据集并对其进行适当转换，类似于之前我们所做的那样。此处的目标是根据学生的特征预测其饮酒程度。这一类特征均为分类值，且高低程度不一而同。图 6.8 显示了相关库和数据集的加载过程。



```
In [1]: import pandas as pd
        import numpy as np
        %matplotlib inline

In [3]: student = pd.read_csv("../data/student/student.csv", sep=";")
        student.rename(columns={'sex':'gender'}, inplace=True)
        student['alcohol_index'] = (5*student['Dalc'] + 2*student['Walc'])/7
        # Alcohol consumption level
        student['acl'] = student['alcohol_index'] <= 2
        student['acl'] = student['acl'].map({True: 'Low', False: 'High'})

In [4]: student.head(3)

Out[4]:
```

	school	gender	age	address	famsize	Pstatus	Medu	Fedu	Mjob	Fjob	...	goout	Dalc	Walc	health	absences	G1	G2	G3	alcohol_index	ac
0	GP	F	18	U	GT3	A	4	4	at_home	teacher	...	4	1	1	3	4	0	11	11	1.000000	Lov
1	GP	F	17	U	GT3	T	1	1	at_home	other	...	3	1	1	3	2	9	11	11	1.000000	Lov
2	GP	F	15	U	LE3	T	1	1	at_home	other	...	2	2	3	3	6	12	13	12	2.285714	Higl

3 rows × 35 columns

图 6.8

针对当前分类模型，图 6.9 显示了所用的全部特征列表。

```
In [5]: features = ['gender','famsize','age','studytime','famrel','goout','freetime','G3']
        target = 'acl'
```

图 6.9

注意：

scikit-learn 库仅支持数字，因此在数字转换过程中，这一点十分重要。对此，可使用称之为独热编码的虚拟特征。

当对变量进行独热（one-hot）编码时，女性学生对应于 0，而男性学生对应于 1。对于 famsize 和 acl（酒精消耗水平），将采用相同的转换操作，对应代码如图 6.10 所示。

```
In [6]: # For gender: Female will be 0, Male will be 1
        student['gender'] = student['gender'].map({'F':0, 'M':1}).astype(int)
        # For famsize: 'LE3' - less or equal to 3 will be 0. 'GT3' - greater than 3 will be one
        student['famsize'] = student['famsize'].map({'LE3':0, 'GT3':1}).astype(int)
        # for acl: 'Low' will be 0, 'High' will be 1
        student['acl'] = student['acl'].map({'Low':0, 'High':1}).astype(int)
```

图 6.10

随后，可将此类数值保存至对象 x 和 y 中，对应代码如下所示。

```
x = student[features].values
y = student[target].values
```

当构建简单模型时，需要预测最为常见的类别。在当前示例中，可通过如图 6.11 所示的代码计算最为常见的类别。

```
In [8]:  student['acl'].value_counts(normalize=True)

Out[8]:  0    0.744222
         1    0.255778
         Name: acl, dtype: float64
```

图 6.11

可以看到，大约 74%的学生的酒精摄入量较低。因此，我们可以构建一个简单的模型，并对 74%的不可见情形进行正确的分类。这一数字十分重要——它提供了第一个基准，随后可以此来比较所构建模型的优良程度。

下面将构建一个称之为逻辑回归的预测模型，导入该模型，创建该对象的实例，并于随后利用我们的数据训练该模型。图 6.12 所示的代码块描述了这一过程。

```
In [9]:  from sklearn.linear_model import LogisticRegression

In [10]: student_classifier_logreg = LogisticRegression(C=2)

In [11]: student_classifier_logreg.fit(X, y)

Out[11]: LogisticRegression(C=2, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

图 6.12

当整体评估当前模型时，需要利用交叉验证对其进行评估。目前，这一评估过程还较为基础，以使我们对这一概念有一个大致的了解，如图 6.13 所示。

```
In [12]: student['predictions_logreg'] = student_classifier_logreg.predict(X)

In [13]: confusion_matrix = pd.crosstab(student['predictions_logreg'], student['acl'])
         confusion_matrix

Out[13]:
```

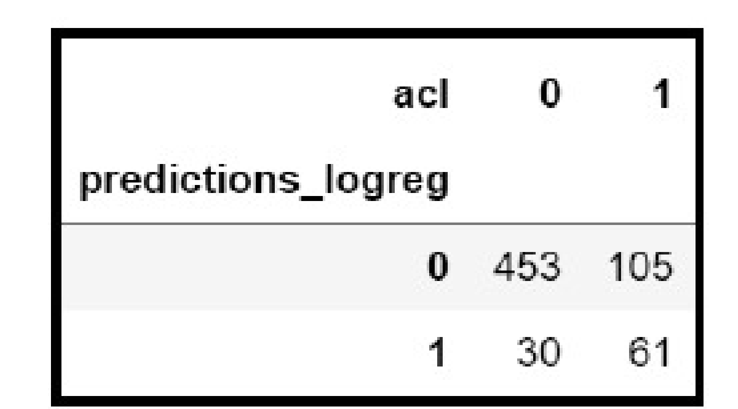
acl	0	1
predictions_logreg		
0	453	105
1	30	61

图 6.13

首先，可计算单元格中的预测结果（计算当前模型生成的预测结果）；随后，将预测结果和实际观察结果制成一个表格，进而构建一个混淆矩阵，如图 6.14 所示。

矩阵的对角线表示分类器做出正确预测的数量，随后可根据该数字计算名为 Accuracy

的简单的评估矩阵。这只是模型所生成的正确预测的一部分内容。在所有这些预测结果中，左上角和右下角对应的数值是正确的结果。



acl	0	1
predictions_logreg		
0	453	105
1	30	61

图 6.14

据此，可计算当前模型的精确度，对应代码如图 6.15 所示。

```
In [14]: ac = (confusion_matrix.ix[0,0] + confusion_matrix.ix[1,1])/student.shape[0]
         print("Accuracy: {}".format(ac))

         Accuracy: 0.7919876733436055
```

图 6.15

通过观察可知，当前模型的精确度为 0.79 或 79%。相应地，可将该值与简单模型的精确度（74%）进行比较。可以发现，两个模型中所得到的数值相差无几，其原因在于，逻辑回归模型是一类十分简单的模型。

另外，我们还可尝试使用更为复杂的模型，即 RandomForestClassifier，并将其作为黑盒以查看精确度方面的变化。针对于此，首先可导入一个对象，创建该对象实例，利用相关数据训练该模型，并于随后生成预测结果，对应代码如图 6.16 所示。

```
In [14]: from sklearn.ensemble import RandomForestClassifier

In [15]: student_classifier_rf = RandomForestClassifier()

In [16]: student_classifier_rf.fit(X,y)
         student['predictions_rf'] = student_classifier_rf.predict(X)

In [17]: confusion_matrix = pd.crosstab(student['predictions_rf'], student['acl'])
         confusion_matrix

Out[17]:
```

acl	0	1
predictions_rf		
0	480	19
1	3	147

```
In [18]: ac = (confusion_matrix.ix[0,0] + confusion_matrix.ix[1,1])/student.shape[0]
         print("Accuracy: {}".format(ac))

         Accuracy: 0.9661016949152542

In [19]: # ['gender', 'famsize', 'age', 'studytime', 'famrel', 'goout', 'freetime', 'G3']
         new_student = np.array([[0, 1, 18, 2, 1, 5, 5, 16]])
         prediction = student_classifier_rf.predict(new_student)
         print("The model predicts that the student belongs to the:")
         if prediction == 1:
             print("HIGH Alcohol Consumption group")
         else:
             print("LOW Alcohol Consumption group")
```

图 6.16

此处构建了一个新的混淆矩阵，如图 6.17 所示。通过观察可知，模型的精确度较高。

acl	0	1
predictions_rf		
0	480	19
1	3	147

图 6.17

最终，模型的精确度为 96%，其原因在于，随机森林这一类复杂模型往往会产生过拟合。过拟合这一概念是指，模型了解数据集中所发生的状况，但未将这一知识一般化至不可见数据。

这也是模型评估本质上非常复杂并且需要交叉验证的主要原因之一。然而，精度矩阵是预测模型中经常使用的一个矩阵。

目前，我们可使用该模型对不可见数据进行预测。假设出现了一名包含新特征的学生，该学生为男性，年龄为 18 周岁，来自于一个多人口家庭；另外，他每周学习两个小时，且家庭关系较差。根据如图 6.18 所示的代码，可预测该名学生的酒精摄入量。

```
In [20]: # ['gender', 'famsize', 'age', 'studytime', 'famrel', 'goout', 'freetime', 'G3']
         new_student = np.array([[1, 1, 18, 2, 1, 5, 5, 10]])
         prediction = student_classifier_rf.predict(new_student)
         print("The model predicts that the student belongs to the:")
         if prediction == 1:
             print("HIGH Alcohol Consumption group")
         else:
             print("LOW Alcohol Consumption group")

         The model predicts that the student belongs to the:
         HIGH Alcohol Consumption group
```

图 6.18

随后，当前模型预测该名学生属于 HIGH Alcohol Consumption group。

对于另一个示例，其中，学生为女性，且在学校表现优异，该学生的期末成绩为 16 分。图 6.19 显示了代码的输出结果。

```
In [19]: # ['gender', 'famsize', 'age', 'studytime', 'famrel', 'goout', 'freetime', 'G3']
         new_student = np.array([[0, 1, 18, 2, 1, 5, 5, 16]])
         prediction = student_classifier_rf.predict(new_student)
         print("The model predicts that the student belongs to the:")
         if prediction == 1:
             print("HIGH Alcohol Consumption group")
         else:
             print("LOW Alcohol Consumption group")

         The model predicts that the student belongs to the:
         LOW Alcohol Consumption group
```

图 6.19

根据上述特征，模型预测该学生属于 LOW Alcohol Consumption group。

6.4　利用回归模型预测房屋价格

本节将利用房产数据集构建回归模型。首先，需要加载房屋价格数据集以供建模使用。随后，将训练一个线性回归模型，并通过一种简单、直观的方式评估该模型，进而通过该模型预测结果。

下面加载相关库并导入数据集。如前所述，我们意识到这样一个事实，在该数据集中，一些社区包含较少的观测结果。为了解决这一问题，仅对超过 30 个观测结果的社区使用当前模型。对此，应使用以下代码块。

```
counts = housing['Neighborhood'].value_counts()
more_than_30 = list(counts[counts>30].index)
housing = housing.loc[housing['Neighborhood'].isin(more_than_30)]
```

根据之前讨论的探索性数据分析，可针对模型选取以下特征。

```
features = ['CentralAir', 'LotArea', 'OverallQual', 'OverallCond',
            '1stFlrSF', '2ndFlrSF', 'BedroomAbvGr', 'Age']
target = 'SalePrice'
```

通过观察可知，目标变量为房屋的 SalePrice；鉴于这是一个数字，因而此处将对回归问题进行处理。

相比之下，Neighborhood 和 CentralAir 并非是数字数据，因而需要转换为数值变量。针对于此，可使用以下代码行。

```
# Neighborhood
dummies_nb = pd.get_dummies(housing['Neighborhood'],
                            drop_first=True)
housing = pd.concat([housing, dummies_nb], axis=1)
# CentralAir
housing['CentralAir'] = housing['CentralAir'].map({'N':0,
                                                   'Y':1}).astype(int)
```

上述代码将针对每个社区生成新的变量，即虚拟变量（dummy variable）。实际上，这针对每个社区创建了包含值为 1 的新向量——房屋属于该社区；否则，将创建一个包含值为 0 的新向量。另外，对于未配置 CentralAir 的房屋，可向其赋予值为 0；否则赋予值为 1。

接下来，向特征列表中加入一些新的特征，并创建包含特征、目标变量和观测结果

的对象，如图 6.20 所示。

```
In [6]: features += list(dummies_nb.columns)

In [7]: X = housing[features].values
        y = housing[target].values
        n = housing.shape[0]
```

图 6.20

在训练模型之前，应确保该模型足够简单。对于回归模型，一种最为简单的模型是预测平均值。对此，可使用如图 6.21 所示的代码。

```
In [8]: y_mean = np.mean(y)
        y_mean

Out[8]: 180167.63358778626
```

图 6.21

对于当前目标变量，平均值约为 180000 美元。针对模型评估，可使用均方根误差评估测量方案，常称作 RMSE，该公式为

$$\mathrm{RMSE}=\sqrt{\frac{\sum(obs-pred)^2}{n}}$$

这里，将观测值与预测值进行比较会发现，观测值和预测值越接近，度量结果就越小。此处我们希望得到尽可能小的度量结果。

当对简单模型计算均方根误差时，可采用如图 6.22 所示的代码行。

```
In [9]: RMSE_null_model = np.sqrt(np.sum((y - y_mean)**2) / n)
        RMSE_null_model

Out[9]: 78032.944854541085
```

图 6.22

经观察可知，对应值约为 70000 美元。

当构建回归模型时，首先需要导入一个对象，并于随后创建该对象的实例，进而训练模型并进行预测，如图 6.23 所示。

```
In [10]: from sklearn.linear_model import LinearRegression

In [11]: regressor = LinearRegression()

In [12]: regressor.fit(X, y)
Out[12]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)

In [13]: housing['predictions'] = regressor.predict(X)

In [14]: y_pred = housing['predictions'].values
```

图 6.23

当计算模型的均方根误差时，需要实现如图 6.24 所示的代码行。

```
In [15]: RMSE_regressor = np.sqrt(np.sum((y - y_pred)**2) / n)
         RMSE_regressor

Out[15]: 33729.218173366113
```

图 6.24

当前模型的输出结果约为 33000 美元，与我们的简单模型相比，这是一个非常低的数字。当在 predictions 与房屋的实际 SalePrice 之间进行比较时（通过可视化方式），可采用如图 6.25 所示的函数生成一个散点图。

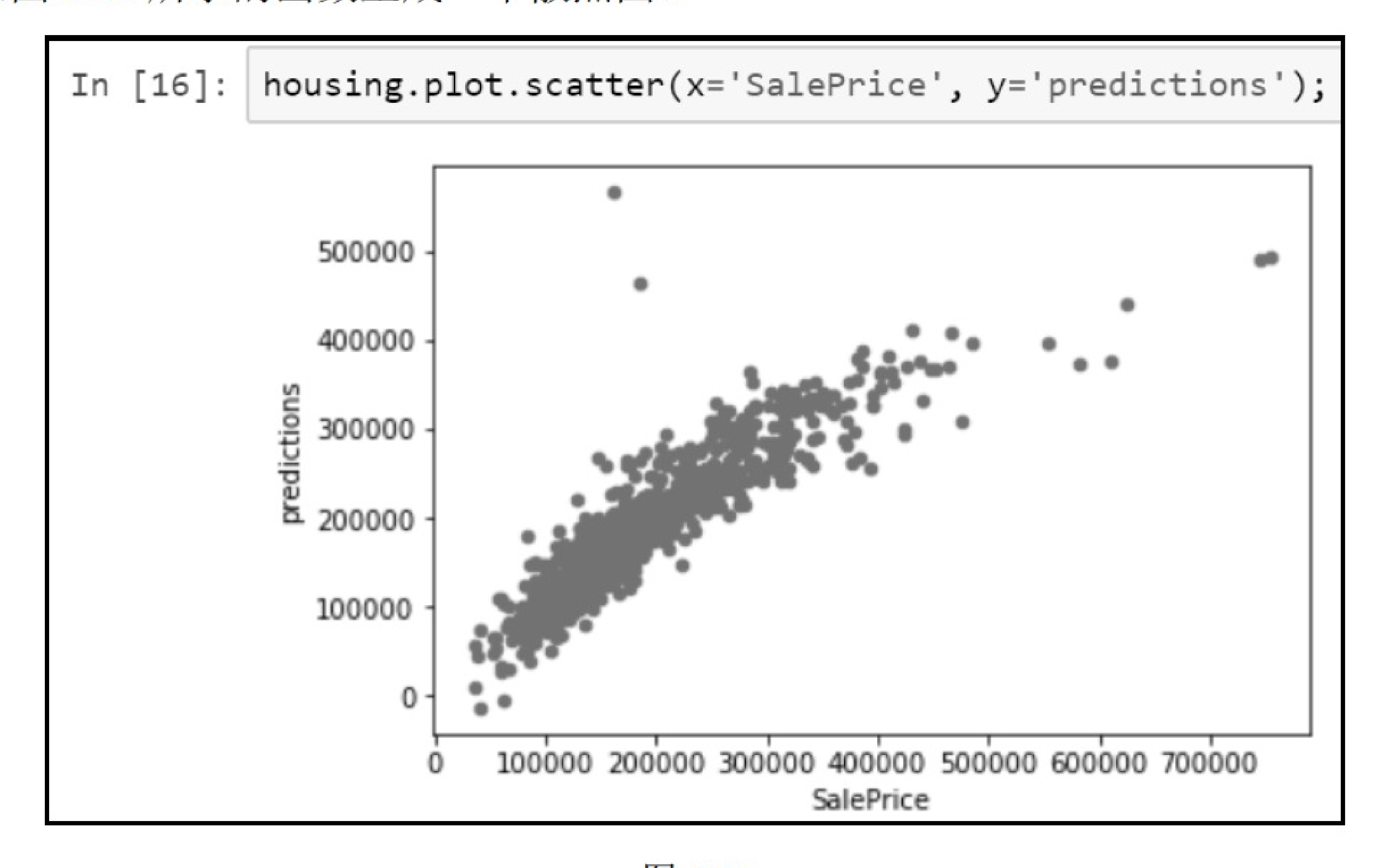


图 6.25

从图 6.25 所示的散点图中可以清晰地看到，预测结果非常接近于房屋的实际

SalePrices，这也说明当前模型具有一定的精确度。

接下来，我们将对包含各类新特征的房屋进行预测。如房屋包含 CentralAir、LotArea、OverallQual（6）和 OverallCond（6）。图 6.26 所示的代码将对房屋价格进行预测。

```
In [17]: new_house = np.array([[0, 12000, 6, 6, 1200, 500, 3, 5, 0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0]])
         prediction = regressor.predict(new_house)
         print("For a house with the following characteristics:\n")
         for feature, feature_value in zip(features, new_house[0]):
             if feature_value > 0:
                 print("{}: {}".format(feature, feature_value))
         print("\nThe predictied value for the house is: {:,}".format(round(prediction[0])))

         For a house with the following characteristics:

         LotArea: 12000
         OverallQual: 6
         OverallCond: 6
         1stFlrSF: 1200
         2ndFlrSF: 500
         BedroomAbvGr: 3
         Age: 5
         Edwards: 1

         The predictied value for the house is: 184,395.0
```

图 6.26

预测结果表明，房屋来自 Edwards 社区，且房价为 184395 美元。

对于不同的示例，可适当地调整社区。假设服务未配置 CentralAir 且属于 Timber 社区，对应代码的输出结果如图 6.27 所示。

```
In [18]: new_house = np.array([[0, 12000, 6, 6, 1200, 500, 3, 5, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1]])
         prediction = regressor.predict(new_house)
         print("For a house with the following characteristics:\n")
         for feature, feature_value in zip(features, new_house[0]):
             if feature_value > 0:
                 print("{}: {}".format(feature, feature_value))
         print("\nThe predictied value for the house is: {:,}".format(round(prediction[0])))

         For a house with the following characteristics:

         LotArea: 12000
         OverallQual: 6
         OverallCond: 6
         1stFlrSF: 1200
         2ndFlrSF: 500
         BedroomAbvGr: 3
         Age: 5
         Timber: 1

         The predictied value for the house is: 214,944.0
```

图 6.27

随后，模型预测房价为 214944 美元。针对不同的房屋，我们可通过相同的代码生成预测结果。需要记住的是，本节所构建的模型并未进行全方位的评估。虽然我们采用了较为常见的评估度量方案，但机器学习模型中的核心步骤一般采用交叉验证。对于模型

的性能评估来说，这一点不可或缺。

6.5 本章小结

本章讨论了预测分析以及与监督式机器学习相关的一些概念，并介绍了如何通过 Python 执行机器学习方面的操作。此外，本章还利用 scikit-learn 考查了预测分析中的各种实例，以及如何训练分类模型以进行预测。最后，本章构建了一个回归模型并以此进行预测。